# NAVAL POSTGRADUATE SCHOOL
## Monterey, California


# THESIS

ALGORITHMIC STUDY ON
SYSTOLIC ARRAY STRUCTURES

by

Leopoldo Jorge de Souza

June 1985

Thesis Advisor: Chin-Hwa Lee

Approved for public release; distribution is unlimited

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

# REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

| | |
|---|---|
| 1. REPORT NUMBER | 2. GOVT ACCESSION NO. |
| 3. RECIPIENT'S CATALOG NUMBER | |
| 4. TITLE *(and Subtitle)*: Algorithmic Study on Systolic Array Structures | 5. TYPE OF REPORT & PERIOD COVERED: Master's Thesis; June, 1985 |
| | 6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(s): Leopoldo Jorge de Souza | 8. CONTRACT OR GRANT NUMBER(s) |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School, Monterey, California 93943 | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS |
| 11. CONTROLLING OFFICE NAME AND ADDRESS: Naval Postgraduate School, Monterey, California 93943 | 12. REPORT DATE: June 1985 |
| | 13. NUMBER OF PAGES: 195 |
| 14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)* | 15. SECURITY CLASS. *(of this report)*: UNCLASSIFIED |
| | 15a. DECLASSIFICATION/ DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution is unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Systolic array, parallel processing, computer simulation

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

Computation bound problems impose a severe burden on the CPU. In order to speed up computation, specific problems that are identified as the main burden can be done using parallel processing. In this way, the time consuming tasks can be executed on specially tailored hardware. This hardware is designed to implement an algorithm-oriented parallel-processing structure that works more efficiently

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S N 0102-LF-014-6601



SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

than the CPU for these specific tasks. This thesis is a study of the mapping of the algorithms onto a kind of structure called systolic array. The development and utilization of a software tool designed to assist on such analysis is presented here. This tool, named Systolic Array Graphics Simulator (SYSGRAS), has the capability to represent any type of systolic array, no matter how complex the cells and structure are. Because of the capability of SYSGRAS, an interactive computer program simulator, the study of systolic arrays is simplified. The complexity of the time-space relationships is analysed with the help of some color graphics techniques. The visualization of the data interaction is thus enhanced and the user is alleviated from the burden of keeping track of partial results and can dedicate attention to the processing algorithm.

# Algorithmic Study on Systolic Array Structures

by

Leopoldo Jorge de Souza
Capitao de Corveta, Marinha do Brasil
Engenheiro Eletricista, Universidade de Sao paulo, 1975
Oficial da Marinha do Brasil, Escola Naval, 1970

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN EIECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 1985

## ABSTRACT

Computation bound problems impose a severe burden on the CPU. In order to speed up computation, specific problems that are identified as the main burden can be done using parallel processing. In this way, the time consuming tasks can be executed on specially tailored hardware. This hardware is designed to implement an algorithm-oriented parallel-processing structure that works more efficiently than the CPU for these specific tasks. This thesis is a study of the mapping of the algorithms onto a kind of structure called systolic array. The development and utilization of a software tool designed to assist on such analysis is presented here. This tool, named Systolic Array Graphics Simulator (SYSGRAS), has the capability to represent any type of systolic array, no matter how complex the cells and structure are. Because of the capability of SYSGRAS, an interactive computer program simulator, the study of systolic arrays is simplified. The complexity of the time-space relationships is analyzed with the help of some color graphics techniques. The visualization of the data interaction is thus enhanced and the user is alleviated from the burden of keeping track of partial results and can dedicate attention to the processing algorithm.

TABLE OF CONTENTS

## LIST OF TABLES

## LIST OF FIGURES

# I. THE DEFINITION OF THE PROBLEM

## A. A FIRST INSIGHT

> "A systolic system is a network of processors which rhythmically compute and pass data through the system. Physiologists use the word "systole" to refer to the rhythmically recurrent contraction of the heart and arteries which pulses blood through the body. In a systolic computing system, the function of a processor is analogous to that of the heart. Every processor regularly pumps data in and out, each time performing some short computation, so that a regular flow of data is kept in the network."
>
> "...matrix computations can be pipelined elegantly and efficiently on systolic networks having an array structure."
>
> "...special purpose hardware devices based on systolic arrays can be built inexpensively using the VLSI technology."

These are parts of the abstract of the paper where Professors H.T. Kung and C.E. Leiserson have first presented the concept of systolic arrays [Ref. 1]. They fully describe the concept and context to which the term "systolic array" applies.

Several authors have presented papers on systolic arrays in recent years, but this subject is very new and everything dates back to 1978.

## B. SYSTOLIC ARRAYS: THE BASIC PRINCIPLE

A systolic system consists of a set of interconnected cells, each capable of performing some simple operation.

Information flows between cells in a pipelined fashion, and communication with the outside world occurs only at "boundary cells".

The usual computational tasks are basically divided into two categories--computing bound and input/output (I/O) bound. If the total number of computing operations is much larger than that of I/O operations, then we have a computing bound task. Systolic arrays are useful to speed up this type of task. The array is able to use each input element a number of times thus achieving a high computational throughput without increasing memory bandwidth. [Ref. 2]

## C. DEVISED ARCHITECTURES

A number of architectures have already been proposed for solving the traditional problems. These are linear arrays for performing matrix-vector multiplication and finding solution of triangular linear systems; two-dimensional arrays for matrix-matrix multiplication/addition, least squares solution and LU factorization,etc [Ref. 3].

## D. HARWARE EXPLORATORY DEVELOPMENT

Because this subject is new, much of the development is still in an exploratory stage. In order to further evaluate the hardware potential of systolic arrays, IRW/ESI has developed a processor called Phoenix that has already been used to provide two-dimensional FIR filters for image processing [Ref. 3: page 3]. To evaluate systolic algorithms and architectures, the Naval Ocean System Center has developed a reconfigurable one-two dimensional systolic array with 64-processing elements [Ref. 4]. This systolic testbed processor can be reconfigured under software control to perform as a rectangular, hexagonal, or linear systolic array. The main goal of its design was to achieve

flexibility in algorithm evaluation, and its cells are much slower than those used in the TRW/ESL Phoenix processor or a possible implementation in a VISI chip.

## E. WHY NOT TO EXPLORE IT WITH SOFTWARE?

During the preliminary investigations in this field, the approach was to select a case study and try to go deeper in the process of understanding the way a particular algorithm is mapped onto a systolic architecture. It was discovered that it is not always straightforward to understand the mapping process. An algorithm like the one proposed by H.T. Kung and W.M. Gentleman for doing matrix triangularization can be really difficult to understand if one does not possess an adequate tool [Ref. 7: page 22]. The algebraic manipulation can be difficult when dealing with systolic arrays because the interaction of the processors tend to lead to a very complicated set of equations in only a few steps and it becomes difficult to achieve any generalization. Some notations have been proposed to help in the modeling the parallel processing [Ref. 5], but no systematic method for mapping an algorithm onto an array exists. There is a lack of the appropriate tools for systolic algorithm development. However, with the development of graphics technology a means now exists that can be used to help people better understand how a systolic array works. If we conjugate this possibility with interactive programming techniques, it may be possible provide an user friendly tool that can be used to achieve the goal which is the main guideline of this investigation: to provide an easier way to understand systolic array operation. A case study will be presented in Chapter III that turned towards performing Matrix Triangularization with the Givens Rotations Algorithm. Eventually, the systolic processing can be

understood and checked against real data with the use of our Systolic Array Graphics Simulator (SYSGRAS). This approach, in contrast with that followed by researchers at the N.O.S.C., in San Diego, is to take full advantage of the software, using computer graphics presentation to improve or to add another dimension to the man/machine interface.

Anyway we want to point out that this work must be seen as an investigation of the possibilities of using this kind of facilities as tools rather than as a final product. Only experimentation in practical work can show the advantages or the limitations of this approach. As in any other area of engineering, feedback from users is important in the evaluation and future improvement of this tool.

## II. THE SIMULATOR APPROACH

The Systolic Array Graphical Simulator (SYSGRAS) is implemented to allow interactive design of any type of systolic array. The cells and their interconnection links are defined by the user in a guided manner. The optional cell types are built into the SYSGRAS. The user can construct any array using the existing cell types or by introducing new cell types, but the introduction of a new cell type cannot be done interactively. This requires a subrcutine to describe and support that type, but the operation is simple and will be explained later. The presently available types can support most of the algorithms found in the literature.

SYSGRAS is designed with a requirement of being user friendly. It has been realized that the amount of data that is normally required to be entered by a user in a session is quite large. It is important to offer the possibility of correcting errors, reviewing entered data, and storing data and results from the session. It can also be used later in another session. The sequence of operations required during a session is shown below:

1) Establishment of general conditions:
   - new or previous problem to run.
   - need to store the calculations for later printing.
   - ability to interrupt or review the progress of systolic processing on a clocked basis.

2) Definition of cells' attributes:
   - type of processor in the cell.
   - graphical shape of the cell.
   - screen coordinates of the cell.

- interfaces (importer cell, receiver of external data; internal cell, only exchanges data with other cells; exporter cell, data transmitter to external world).

3) Definition of links:
   - define the origin and destination of each link. The origin is a port in a cell. Each cell has a number of output and input ports. The function of each one depends on the processor type. The destination is a input port in another cell. A link is defined by origin cell output port and the destination cell input port.

4) Presentation on the graphical screen:
   - the picture of the user defined array is initially presented on the screen with all processor registers initialized to zero. Since the presentation is strongly based on colors, the non existence of data interaction in the cells makes them all turn into black. The identification number given by the user to each cell is placed at the cell's lower right corner and red arrows indicate the linking between cells and the direction of data flux (see Fig. 4.7 at Chapter IV).

5) Data input and screen update:
   - as soon as data originated from the external world to the importer cells is entered, the whole screen is updated and representing the status of the systolic array at that particular clock cycle. The external data are entered at each clock cycle, and the screen is updated accordingly.

6) Review of the ccmplete systolic process:

- optionally, the user can review the complete process. The tctal data generated in a session are stored automatically, and the graphics presentaticn of any previous session can be seen again without any additional data entry.

The central idea of the SYSGRAS is to allow a user to construct any desired systolic array and to use as the input test data such that insight can be gained about the data interactions of the systolic algorithm. The user has the capability to correlate the propagation of the input data with the colors of the cells to better understand the process. Certainly, the greater the creativity of assigning the cclcr, the better the result showing the interactions. There is no restrictien as to how to do the correlatien and several examples will be presented. The colors that are available to a input datum are red, green and blue. If a cell receives data frcm more than one source with different primary colors, these colors will combine. The cell will show cn the screen with a resulting color that is the combination of the input primary colors. This resulting colcr is not passed on to other cells. The propagation of data is associated with the primary colors, and this way a gcod tracking of the timing that takes for each data wavefront to propagate through the array can be achieved. The screen display shows the array in a way selected by the user. The cells are presented with the shape, position, identification and ccnnections requested by the user. The colors and numerical results assumed in each clock cycle show the data interaction. The numerical results that appear on the screen is in a fixed format and with rather limited precision. If one wants greater precision, there is an cptien to

record the session which can show results with up to five decimals on printer cutput. These results can be checked against the known values to verify the correct operation of the mapped systolic algorithm.

## III. A CASE STUDY: MATRIX TRIANGULARIZATION BY GIVENS ROTATIONS

### A. A REASON FOR MATRIX TRIANGULARIZATION: QR DECOMPOSITION

The technique of QR decomposition is useful to solve the Linear least Squares Problem. There are other possible approaches in the literature to solve this type of problem [Ref. 6]. We will select the QR Decomposition Method as mentioned by Gentleman and Kung in [Ref. 7: page 19]. Let's present a brief explanation of the method.

Given a linear system defined by

$$AX = B$$

where A is a (n x p) matrix, X is a (p x 1) and B is a (n x 1) column vector. We want to find the vector X such that $||r||=||B-AX||$ is minimized. The vector X is also called vector of regression coefficients. This is in fact a vector of estimated elements and as such it strictly should be written $\hat{X}$ but for convenience X will continue to be used. Also a more rigorous matrix notation would be, for example, $\underset{\sim}{A}$ instead of A. If we are able to find an orthogonal matrix Q such that QA=R where R is an upper triangular matrix, then we will have

$$QAX = RX = QB$$

and, as a result, RX = QB.

As R, Q, and B are all known, we can find X by Back Substitution. Gentleman has shown, as seen in [Ref. 8: page 329], that this approach solves the Least Squares Problem because it solves the normal equations of the problem [Ref. 6: page 148].

An accurate QR Decomposition can be obtained in several ways, like the Gram-Schmidt Process or Householder Transformations. For this particular study we will stick to the Givens Rotations Method. This method requires the use of a sequence of plane rotations on matrices A and B, that will convert the linear system to a representation of the form

$$RX = QB$$

which is straightforward to solve.

The Givens Rotations Method has two different implementations [Ref. 8: page 331]. We will study the one called "with square roots".

## B. QR DECOMPOSITION BY GIVENS ROTATIONS WITH SQUARE ROOTS

If Q is an orthogonal matrix, Y=QA is called an orthogonal transformation and the columns of Q, say q(1), q(2), ... are orthogonal. In order to better understand what an orthogonal transformation means, we can interpret the q(i)'s as unit vectors (because of their unit length) that represent the new reference frame in the original coordinates frame. If we apply an orthogonal transformation to a matrix (or to a vector), we will be in fact changing the reference coordinates for that vector. The information content of the transformed matrix (or vector) will be the same, although different from previous form because of the new reference. The columns of Q are given by the direction cosines of the new reference axes with respect to the old reference system [Ref. 12: page 354].

An orthogonal transformation can be divided into a sequence of elementary transformations. In the case of Givens Rotations, each one of these elementary transformations is a rotation about one coordinate axis of the original reference frame (which may be n-dimensional).

In geometry we know that the transformation

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} \cos(x1) & \cos(x2) & \cos(x3) \\ \cos(y1) & \cos(y2) & \cos(y3) \\ \cos(z1) & \cos(z2) & \cos(z3) \end{bmatrix} \cdot \begin{bmatrix} a1 \\ b1 \\ c1 \end{bmatrix}$$

applied to a vector $[a1,\ b1,\ c1]^T$ will rotate the original reference frame about the 3 axes at the same time. The direction cosines of the new x-axis are cos(x1), cos(x2) and cos(x3); the direction cosines of the new y-axis are cos(y1), cos(y2) and cos(y3), and so on. The transformation effect achieved by the matrix of cosines is similar to that obtained by the Givens Rotation transformation matrix Q.

We can split the cosines matrix into a product of matrices each one representing a rotation about one axis of the original frame. This technique of decomposing the operation into product operations, each one being responsible to rotate the system matrix of an angle TETA about one axis of the original frame, is exactly what is done in Givens Algorithm. Each individual Givens' Rotation is represented by a matrix of the form

$$D(j,i) = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & 0 & \cdots & & 0 \\ 0 & & & & \cdots & & 0 \\ 0 & \cdots & 0 & c & 0 & \cdots s \cdots & 0 \\ 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \\ 0 & & & & \cdots & & 0 \\ 0 & \cdots & & 0 & 1 & 0 \cdots & 0 \\ 0 & \cdots -s \cdots & 0 & c & 0 & . & 0 \\ 0 & & & & \cdots & 1 . & 0 \\ 0 & & & & \cdots & & 0 \\ 0 & & & & \cdots & 0 & 1 \end{bmatrix} \begin{array}{l} \\ \\ \\ \leftarrow\!= \text{ i th row} \\ \\ \\ \\ \leftarrow\!= \text{ j th row} \\ \\ \\ \end{array}$$

where sqr(c) + sqr(s) = 1, c = cos(TETA), s = sin(TETA) and dots represent zeros [Ref. 6: page 153]. This elementary rotation will convert the (j,i) element of the matrix which it premultiplies intc zero, that is, the (j,i) element of the product of matrix D(j,i) and A will be

$$-s.a(i,i) + c.a(j,i) = 0$$

and sc it follows that

$$d = sqrt(sqr(a(i,i)) + sqr(a(j,i)))$$

$$c = a(i,i)/d \quad \text{and} \quad s = a(j,i)/d$$

If the matrix A is (n x n), it will be necessary tc have (n-1) elementary rotations to turn all elements of the first column ( with exceptien of a(1,1) ) onto zero, (n - 2) rotations for the second column, and so on. For a (3 x 3) matrix, three rotatiens will suffice.

## C. MAPPING GIVENS RCTATIONS AIGORITHM INTO A SYSTOLIC ARRAY

The problem of mapping an algorithm into a systolic structure freguently turns out to be the subject of certain restrictions because the computed results may result in very small numbers impossible to be represented if fixed point hardware is used. In crder to avoid establishing undesirable conditiens on the data, one must try to select an algcrithm possible to be mapped and to perform computations cn any data within the selected range of approximations.

We have already presented the Givens Algorithm. The problem now is to understand how a systolic array can realize it. The reader must be aware that the understanding of the mathematical algorithm does not lead to immediate insight cn how the array works. For the original discussion on this algorithm the reader is referred to [Ref. 7].

Presented in the following are the specifications of the cell processors used in this systolic array, the general cell arrangement, and the input data streams. However, the sight of such specification does not reveal the intricacies of the deduction of which function should be performed by each particular cell element. A possible way to gain insight as to how the algorithm is mapped onto the array is to perform an algebraic validation that is extremelly laborious. For a start, suppose that we want to triangularize a (3 x 3) matrix A. We must premultiply it 3 times by elementary rotation matrices which will turn the elements a(2,1), a(3,1) and a(3,2) into zeros. The first transformation will be

$$\begin{bmatrix} c & s & 0 \\ -s & c & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} a(1,1) & a(1,2) & a(1,3) \\ a(2,1) & a(2,2) & a(2,3) \\ a(3,1) & a(3,2) & a(3,3) \end{bmatrix}$$

This elementary rotation will turn element (2,1) into null on the product matrix. We can note that the element (-S) at the transformation matrix occupies the position of the element that will be turned into zero in the transformed matrix. The transformed matrix will become

$$\begin{bmatrix} a'(1,1) & a'(1,2) & a'(1,3) \\ 0 & a'(2,2) & a'(2,3) \\ a(3,1) & a(3,2) & a(3,3) \end{bmatrix}$$

which will be operated by a second transformation matrix

$$\begin{bmatrix} c & 0 & s \\ 0 & 1 & 0 \\ -s & 0 & c \end{bmatrix} \cdot \begin{bmatrix} a'(1,1) & a'(1,2) & a'(1,3) \\ 0 & a'(2,2) & a'(2,3) \\ a(3,1) & a(3,2) & a(3,3) \end{bmatrix}$$

and that will generate the matrix

$$\begin{bmatrix} a''(1,1) & a''(1,2) & a''(1,3) \\ 0 & a'(2,2) & a'(2,3) \\ 0 & a'(3,2) & a'(3,3) \end{bmatrix}$$

Finally the last transformation

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & c & s \\ 0 & -s & c \end{bmatrix} \cdot \begin{bmatrix} a''(1,1) & a''(1,2) & a''(1,3) \\ 0 & a'(2,2) & a'(2,3) \\ 0 & a'(3,2) & a'(3,3) \end{bmatrix}$$

will generate the upper triangular matrix

$$\begin{bmatrix} a''(1,1) & a''(1,2) & a''(1,3) \\ 0 & a''(2,2) & a''(2,3) \\ 0 & 0 & a''(3,3) \end{bmatrix}$$

It is important to note that the s's and c's are different in each transformation matrix and are calculated the way we have already pointed out, that is, for the first elementary transformation,

$$d = \text{sqrt}(\text{sqr}(a(1,1)) + \text{sqr}(a(2,1)))$$

$$c = a(1,1)/d \quad \text{and} \quad s = a(2,1)/d$$

Now the evaluation of the elements of each transformed matrix should be considered by performing the actual multiplications to correlate the resulting algebra with the algorithmic processor's equations presented in Figs. 3.1 and 3.2. For this particular algorithm it requires definitely a lot of paperwork. This cannot be accepted as a good method if one wants to understand the general data flow in the

systolic array, and to check its numeric data. This is why we develcped the SYSGRAS to have the possibility of doing an evaluaticn study on how several algorithms found in the literature do perform.

## D. A NUMERICAL EXAMPIE

In order to test the Givens Algorithm in doing Matrix Triangularization, we have prepared a numeric study case. A system is represented by the matrix equation

$$\begin{bmatrix} 2 & 4 & 1 \\ 5 & 7 & 4 \\ 3 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x(1) \\ x(2) \\ x(3) \end{bmatrix} = \begin{bmatrix} 12 \\ 3 \\ 8 \end{bmatrix}$$

In Appendix C we shcw how to operate on this equaticn for triangulating and to solve it by Back Substitution. Using these numerical data, we will set up the problem fcr the simulator and be able to check out the simulation results against our hand calculated values.

## E. SETTING UP THE PROBLEM FOR SIMULATION

### 1. Sketching the Graphics

The first step in preparing the problem for simulation consists of planning the graphics. We have tc consider the fcllcwing points:

- How do we want our systolic array arranged on the screen?
- What kind of shape will be used for the cells (in SYSGRAS we have two options: square and octogcnal)?
- The screen coordinates for the cells (SYSGRAS divides the screen into a chessboard, and therefore we have a span of 8x8 positions to place the array cells).

- Which are the input and output ports? This is dependent on the subroutine that implements the cell processor, and so must be compatible with the definition of the processor cell subroutine.
- Which links are used to connect the different ports of the different cells? We need to identify everything so that the correct information at SYSGRAS can be entered.

This may be better understood with an example, and we will continue the development of the systolic array to perform Matrix Triangularization.

For this particular case, we will use three types of processors: a Givens External cell, a Givens Internal cell (both with square roots), and a Buffer cell. The last one is used to help the visualization of the input data array being pumped into the systclic array.

Figures 3.1, 3.2, and 3.3 show which ports are considered the active terminals in the actual implementation

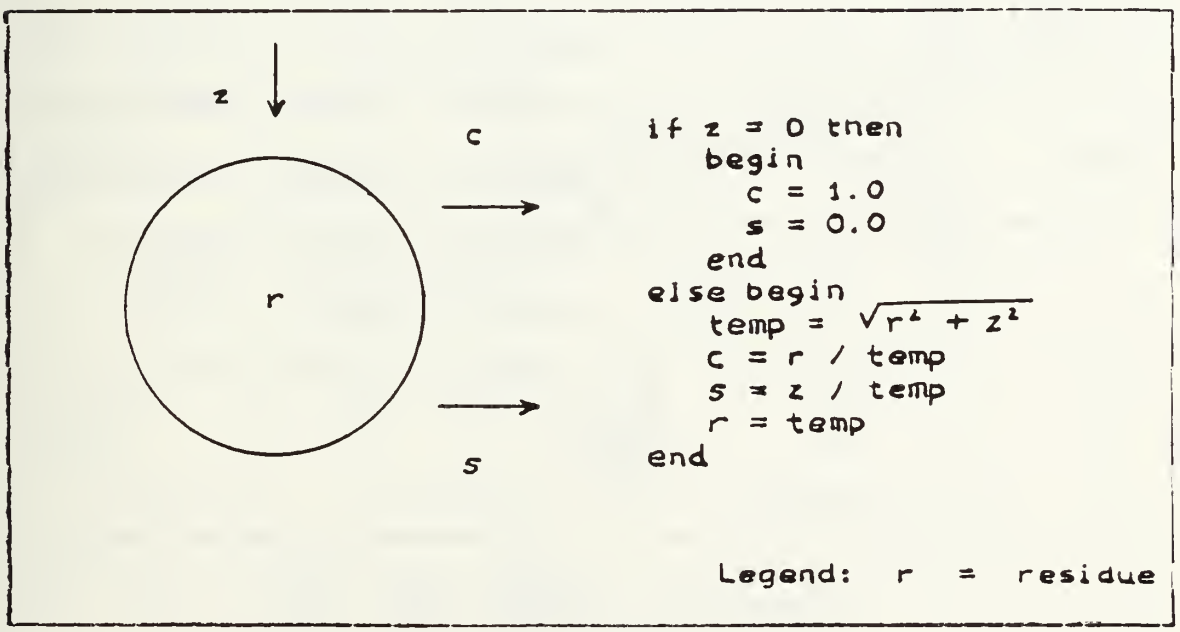



Figure 3.1 Givens External Processor Cell

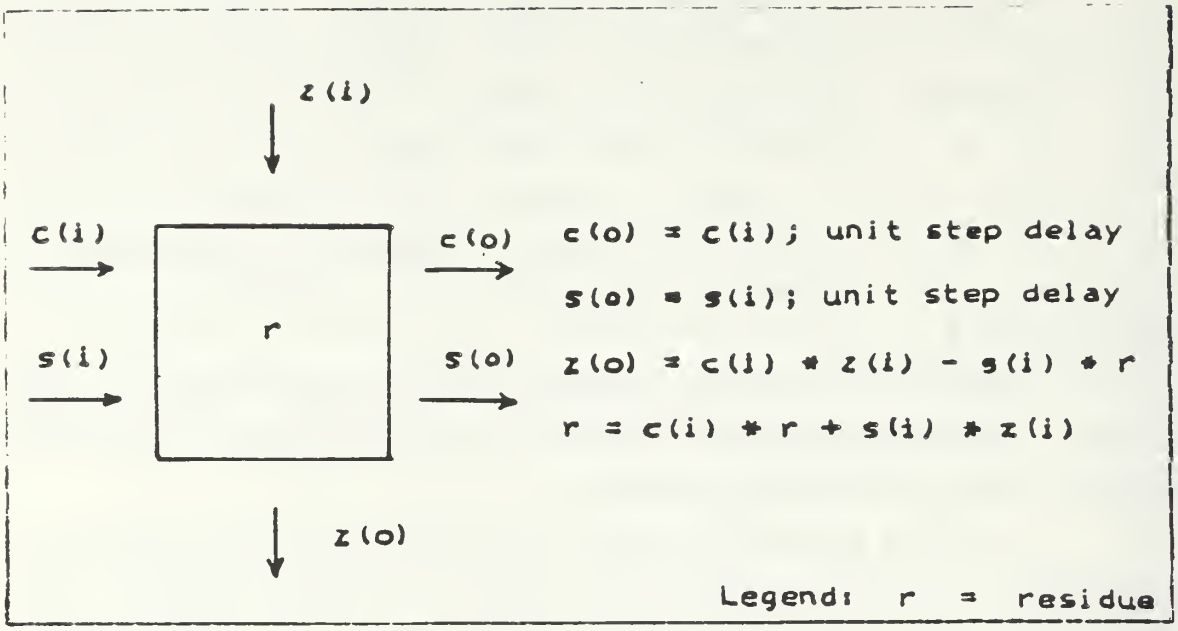



Figure 3.2 Givens Internal Processor Cell

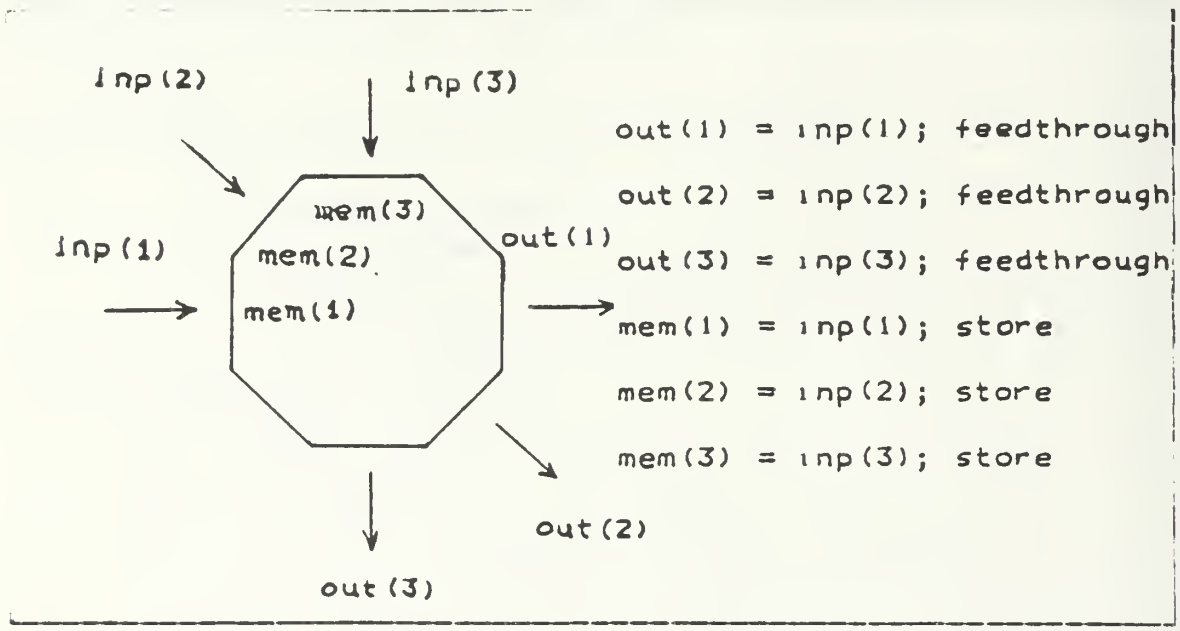



Figure 3.3 Buffer Cell

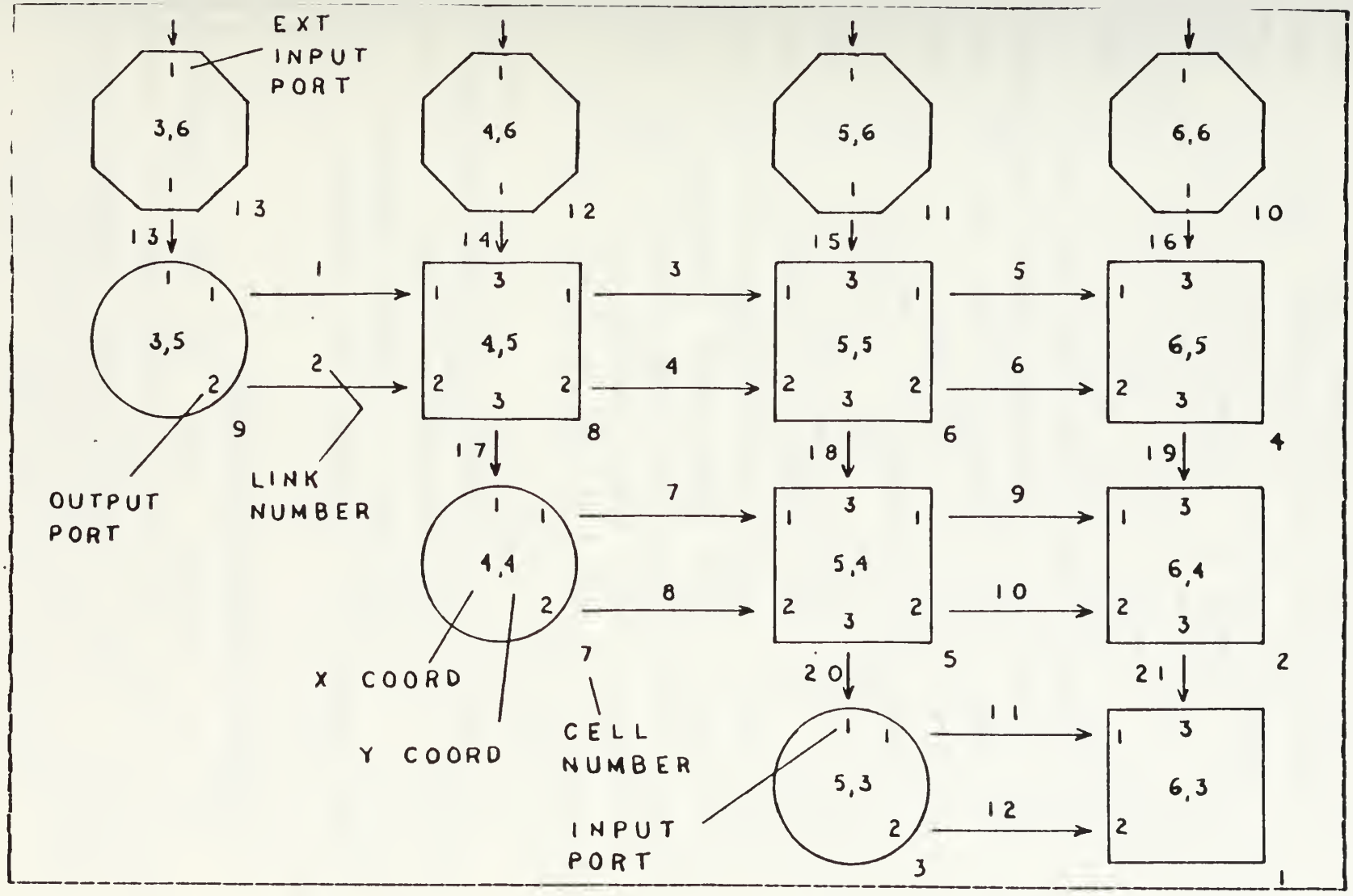



Figure 3.4 Systolic Array for Matrix Triangularization

in SYSGRAS for these cell processors. Fig. 3.4 show how to arrange the cells and their interconnections. It is important to realize to the fact that although circular shaped cells are presented in Figs. 3.1 and 3.4, they are implemented in the SYSGRAS as octogonal shaped cells. This approach was adopted to improve the computational speed of the simulator. However, it has been decided to keep the standard circular shape in Figs. 3.1 and 3.4 . An additional detail for the reader is the fact that cells 10 to 13 in Fig. 3.4, buffer cells, do not show exactly the same way as Fig. 3.3 does. This is because a buffer cell is implemented in SYSGRAS with three channels, but for Matrix Triangularization only one channel is used (and so only one output port is shown in Fig. 3.4). Attention should also be paid to the fact that the arrows shown in Fig. 3.4 represent the links that have actually been used in our simulation. Cells number 1, 2 and 4 of that Figure are of the same type of the other square shaped cells that appear in the same Figure, although there is no arrow drawn on their right side. The reason is that a third order system is used in the simulation and so there is no need to extend the limits of the array beyond those cells.

This type of preparation that has been presented here is very helpful when entering the data because there is a large amount of information asked by SYSGRAS to be entered interactively. If every detail is planned beforehand, the interaction between the user and the simulator becomes faster and easier.

2. Planning the Input Data Array

Once the systolic array has been planned and sketched, the user must prepare the test data that exercise the simulator. This is pehaps the most important step in the simulation. At this point the only rule to follow is

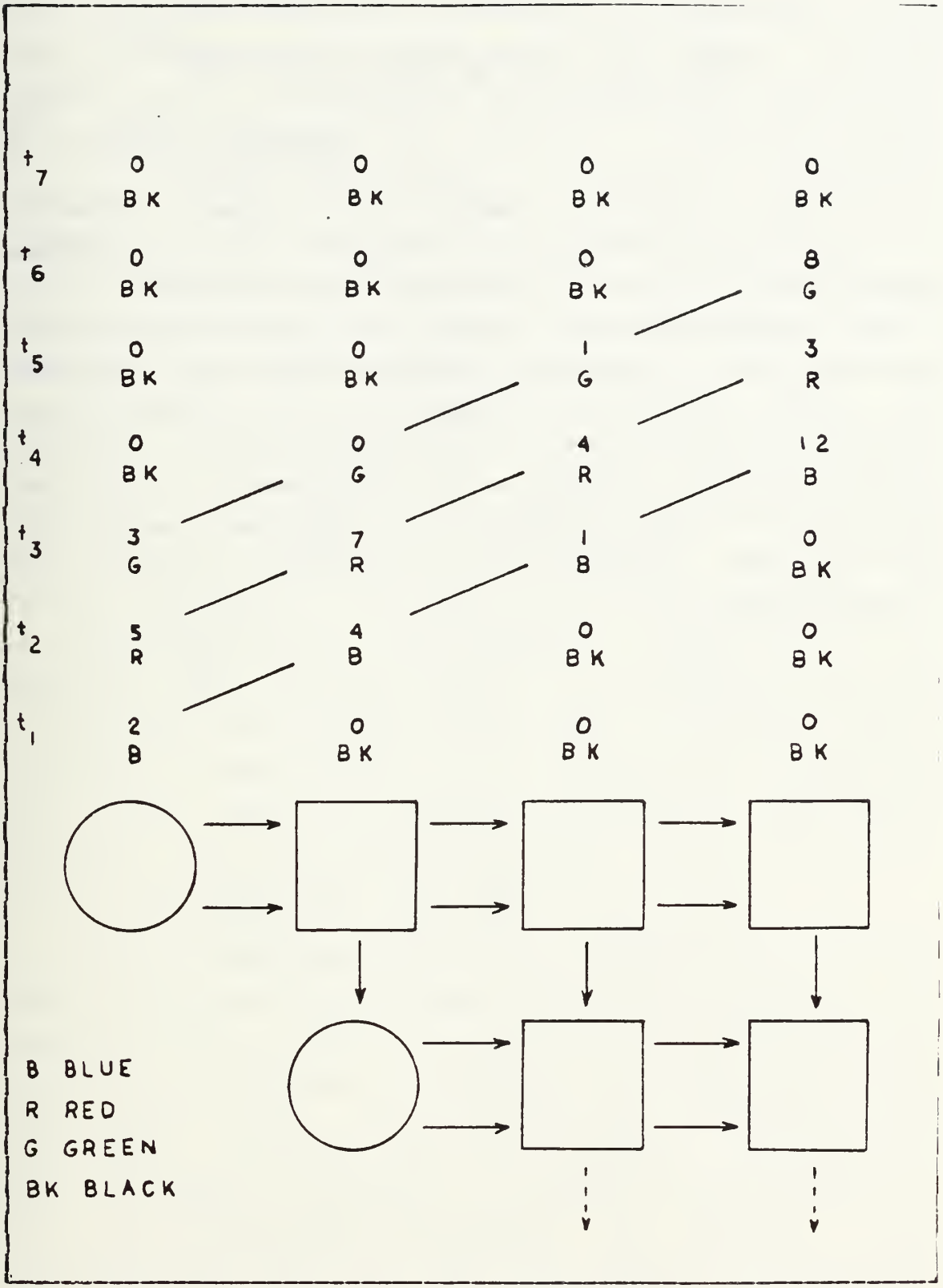



Figure 3.5 Input Data Array Arrangement

that a time shift as required by the algorithm has to be respected. The way color information will be assigned to the entering data is completely up to the user, and the greater his/her creativity, the better the simulation effect, and consequently, the easier to interpret the results.

Fig. 3.5 shows a possible way to prepare the data. We can identify in that figure the numbers from our numeric example referred in last subsection. The time shift that can be seen is necessary to provide the necessary synchronism for the systolic processing. In some algorithms, the output data to the external world must also be collected in a synchronous fashion. But, for the particular case of Matrix Triangularization, as soon as the final result is achieved, each datum is frozen in its cell and waits for a pumping out operation whose connection links are not being considered in our simulation.

# IV. THE ANALYSIS OF THE SIMULATIONS

## A. DIVIDED DIFFERENCES

Before going into detailed analysis of the Matrix Triangularization some simple algorithms are presented.

In numerical interpolation, we need to compute the divided differences from a set of points and then form a polynomial from these divided differences. The problem of mapping the algorithm for calculating these differences into a systolic array structure has been recently addressed in [Ref. 10]. In that paper, Li and Smith propose a systolic architecture consisting of triangular cells. Some upward, some downward, according to that paper, have the same internal architecture but must perform differently depending on their orientation with respect to the data flow. Certainly, this poses the problem of sensing the direction of the data flow and implementing some additional logic in the cells to change their function accordingly. Li and Smith also discuss some implementation problems in their paper, and point out that their actual cell "is not exactly the same as described" [Ref. 10: page 542].

Their algorithm is implemented in SYSGRAS, but the basic cell has been modified; grouping one upward triangle and one downward triangle into the same cell. It is noticed that the upward cell was in fact actuating just as a traffic-orientor buffer and no additional information was being incorporated into the data flow. It wouldn't be cost effective to have that kind of processor since the introduced complexity was due to the need to establish a homogeneous type of architecture based on triangles. Therefore, our basic cell became that shown in Fig. 4.1, and the systolic architecture,

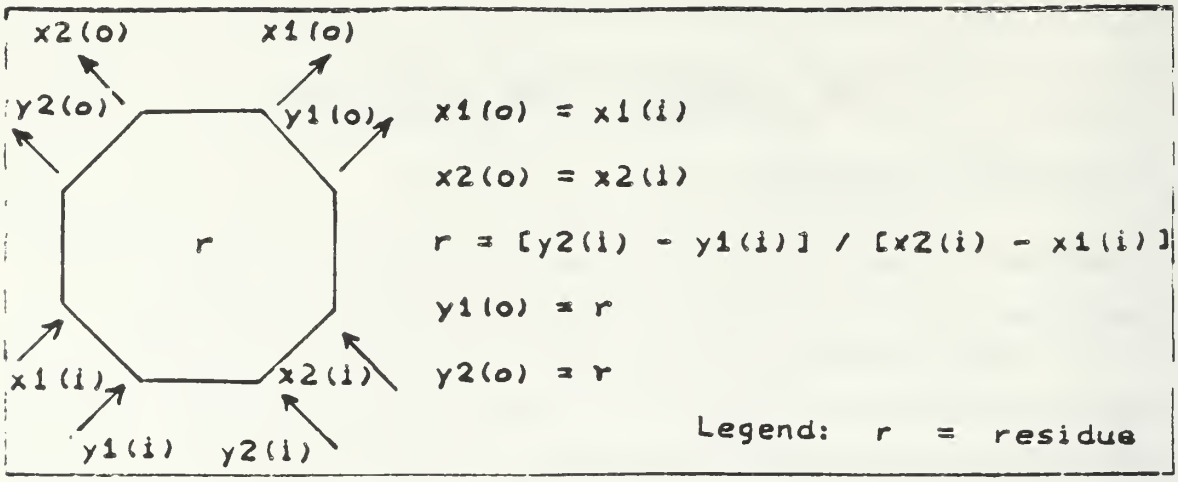



**Figure 4.1 Divided Differences Cell Processor**

almost the same as that proposed by Li and Smith, appears in Fig. 4.2.

We have investigated the problem of finding the divided differences of the set of points (1.0,1.0), (1.8,2.2), (3.0,3.3), (4.3,3.5) and (5.3,4.1) in the x-y plane. According to [Ref. 10: page 539], the first level divided differences are

$$y'(1) = (y(2)-y(1))/(x(2)-x(1)) =$$
$$= (2.2-1.0)/(1.8-1.0) = 1.5000$$
$$y'(2) = (y(3)-y(2))/(x(3)-x(2)) = 0.91666$$
$$y'(3) = (y(4)-y(3))/(x(4)-x(3)) = 0.15385$$
$$y'(4) = (y(5)-y(4))/(x(5)-x(4)) = 0.6000$$

Continuing with the calculations, the second level divided differences are

-0.29167
-0.30512
0.19398

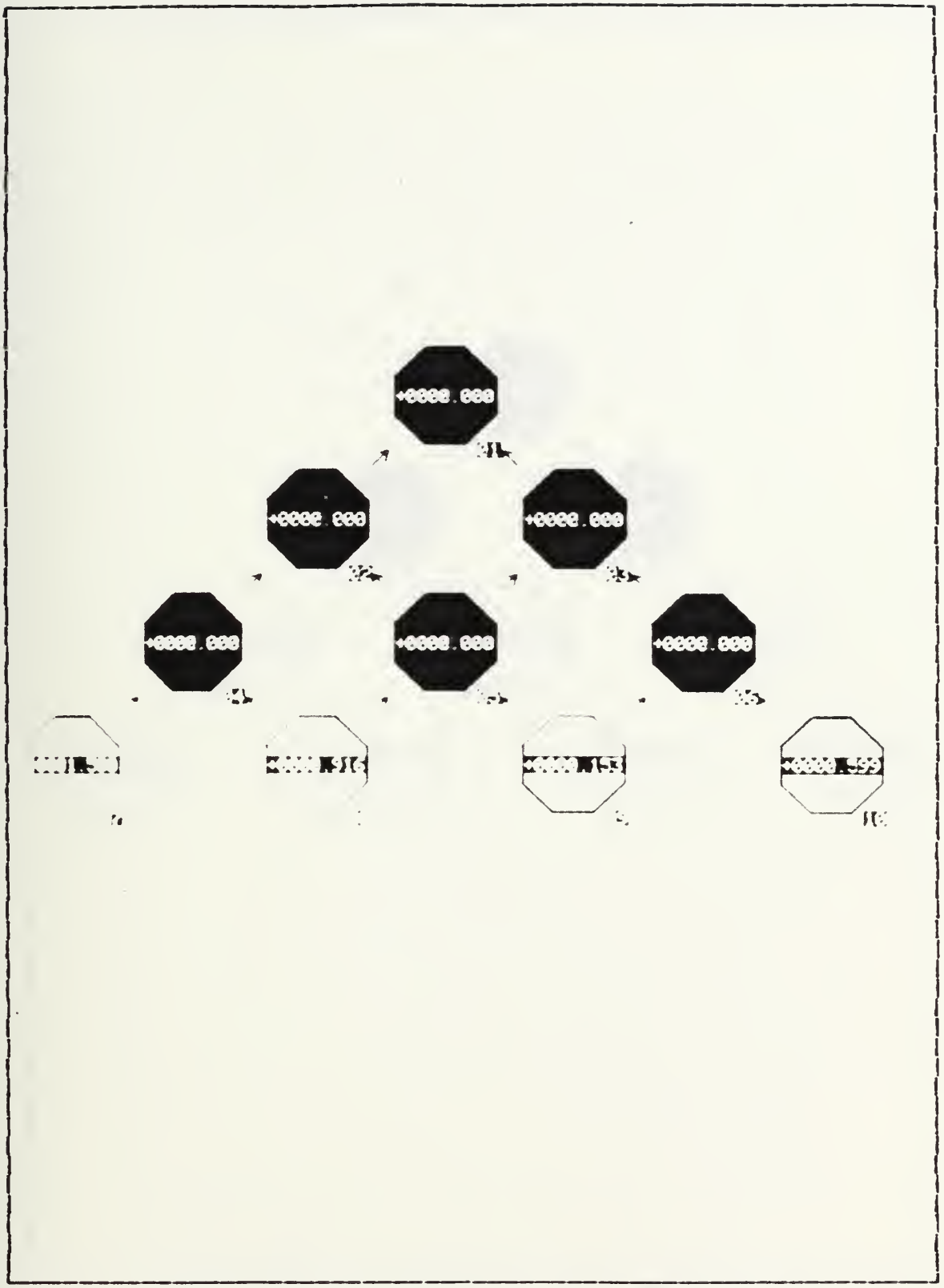



Figure 4.2 Divided Differences Array after Clock Cycle 1

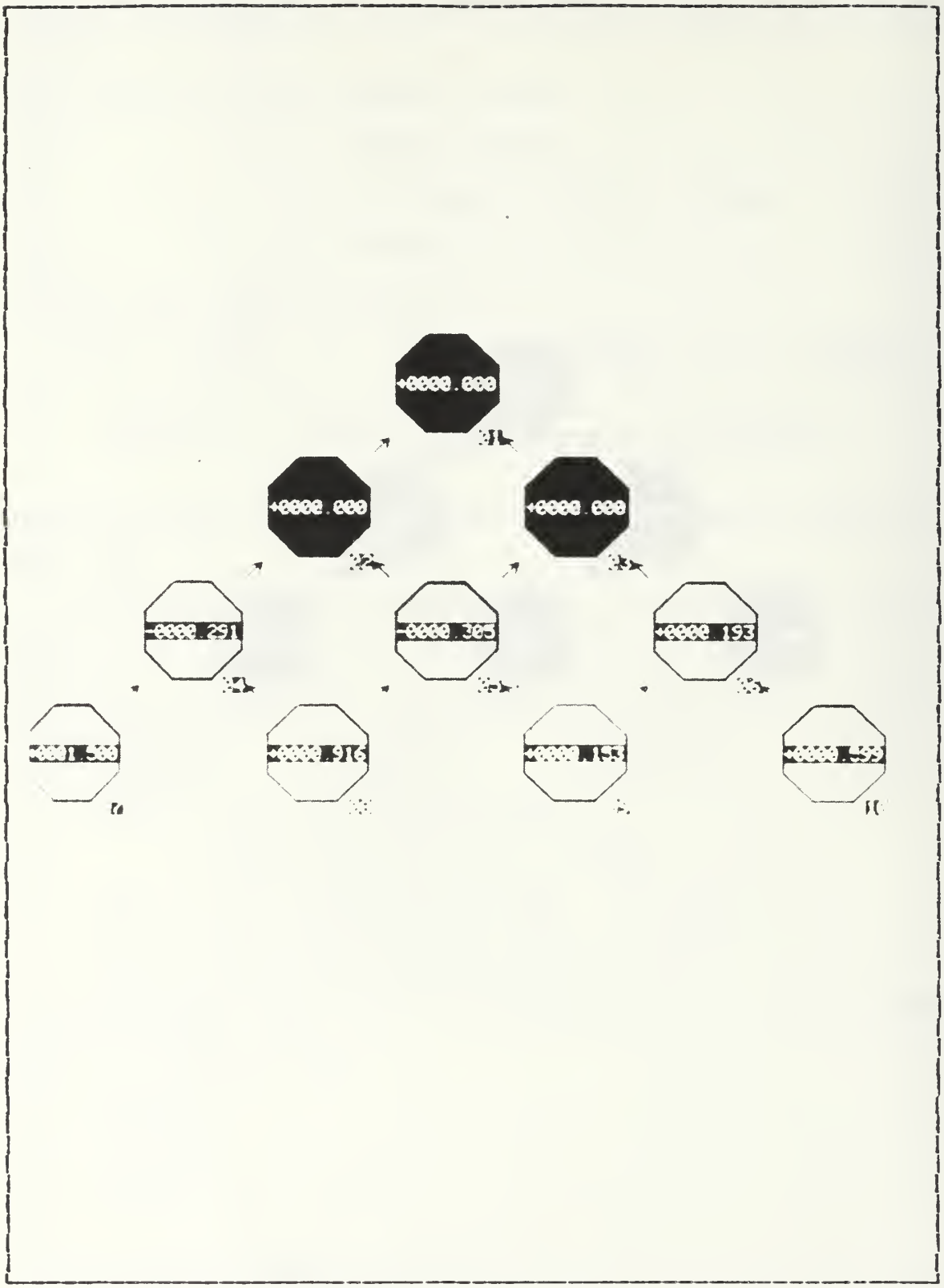


Figure 4.3 Divided Differences Array after Clock Cycle 2

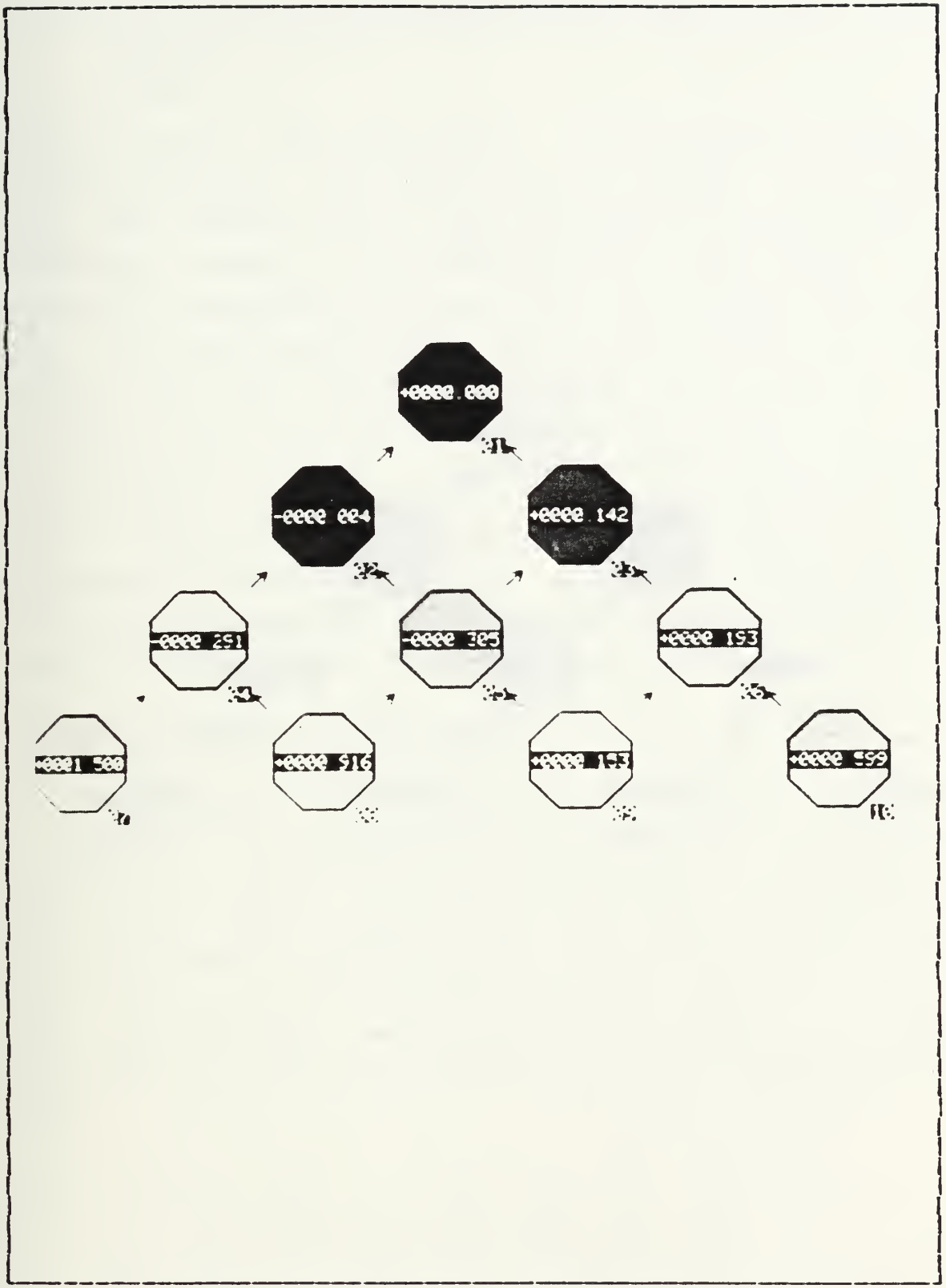



Figure 4.4 Divided Differences Array after Clock Cycle 3

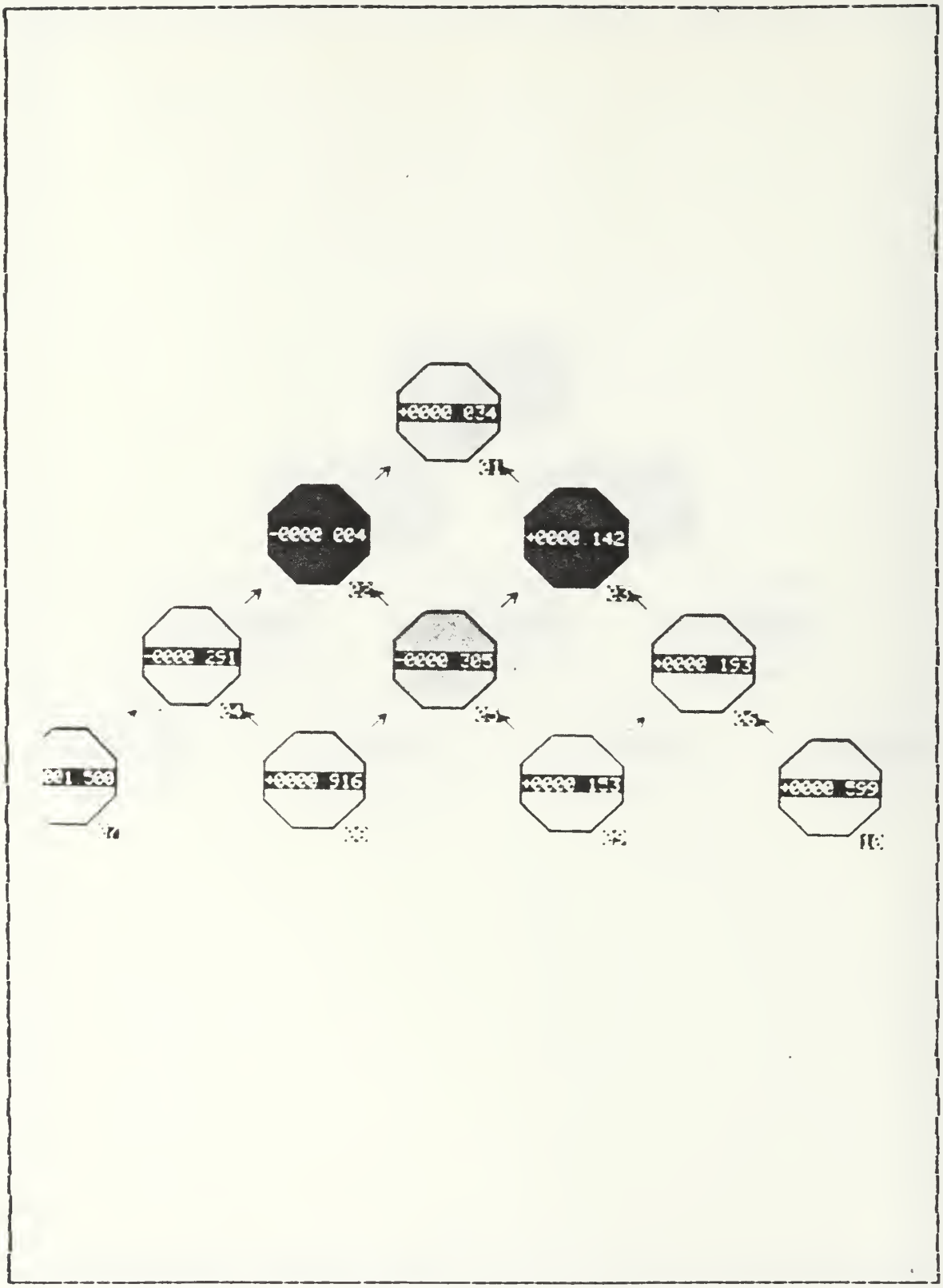



Figure 4.5 Divided Differences Array after Clock Cycle 4

and the third level divided differences are

-0.00408

0.14260

and the fourth level is 0.03411. This suggests a kind of pyramid structure that can be seen from Figs. 4.2 up to 4.5. The calculation of the differences is performed in four-clock cycles, one represented in each figure. The bottom row displaying the first order differences, the upper row, the second order and so on till the top cell. We have related each point (P1-P5) to a primary color, that is, P1 is blue, P2 is red, P3 is green, P4 is blue, and P5 is red. In Fig. 4.2 we can see the result of the first step of calculations, with the mixing of these colors at the bottom cells. Going through each step and comparing them with the mathematical formulation, we get a better understanding of the data interaction. The the final results in Fig. 4.5 should be compared with the above calculated values.

## B. MATRIX-MATRIX MULTIPLICATION

This is another simulation problem that have been tried on SYSGRAS. The presentation of the algorithm has been done at [Ref. 1: page 9]. Presented in Fig. 4.7 is the arrangement of the systolic array and how the data is pumped into it. The elements of matrix A can be seen flowing towards the lower right and those of matrix B flowing towards the lower left. The resultant matrix C is pumped upwards. We have performed a simulation to compute the matrix product AB=C as below

$$\begin{bmatrix} 1 & 2 & 0 \\ 4 & 7 & 1 \\ 8 & 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} 2 & 1 & 9 \\ 3 & 7 & 4 \\ 0 & 10 & 6 \end{bmatrix} = \begin{bmatrix} 8 & 15 & 17 \\ 29 & 63 & 70 \\ 22 & 52 & 98 \end{bmatrix}$$

The scheme used for systolic computation has an advantage due to the fact that a great number of matrices that are used in typical problems are band matrices [Ref. 9]. Consequently, the systolic array does not need to include as many cells as it would have to for the case of full matrices. In this numeric example, in order to restrict our array to a small size so that it can be seen on the screen (as in Fig. 4.8), we decided to set elements a(1,3) and b(3,1) to zero. This way, a systolic array of small dimensions can multiply larger matrices with restricted nonzero band.

A single type of cell is used in this algorithm. It is called Inner Product Step Processor. Its geometry and algorithmic definition are shown in Fig. 4.6 . This same type of cell will be presented later in another algorithm implementation.

In this simulation problem, different colors are attributed to different rows of matrix A and to different columns of matrix B. As we know, each element of the product matrix C will be generated by a combination of one row of A and one column of B. When these elements join each other in a systolic cell, we will be able to identify exactly that combination taking place at each cell in the space-time frame. We present here the coding of colors that was adopted:

$$\begin{bmatrix} \text{blue} & \text{blue} & \text{blue} \\ \text{red} & \text{red} & \text{red} \\ \text{green} & \text{green} & \text{green} \end{bmatrix} \cdot \begin{bmatrix} \text{blue} & \text{red} & \text{green} \\ \text{blue} & \text{red} & \text{green} \\ \text{blue} & \text{red} & \text{green} \end{bmatrix}$$

$$= \begin{bmatrix} \text{blue} & \text{magenta} & \text{cyan} \\ \text{magenta} & \text{red} & \text{yellow} \\ \text{cyan} & \text{yellow} & \text{green} \end{bmatrix}$$

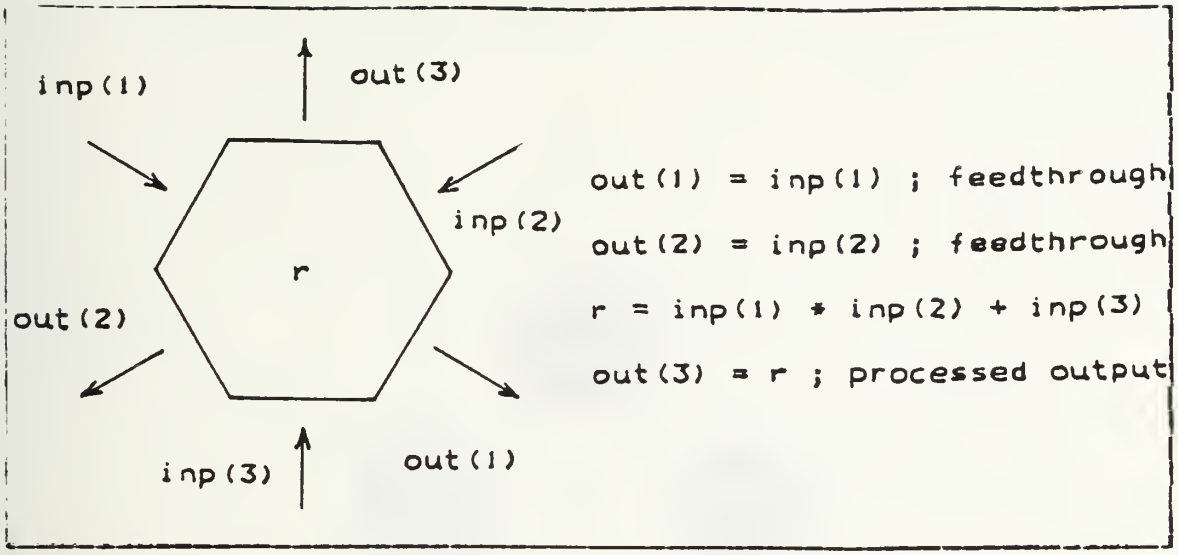



Figure 4.6 Inner Product Step Processor Cell

To show how colors can help in understanding the mechanism of multiplication in the systolic array, we will track the generation of the element c(3,2). We see from the above color coding that c(3,2) is a yellow element, since it results from the combination of a green row and a red column. From mathematics,

$$c(3,2) = a(3,1).b(1,2) + a(3,2).b(2,2) + a(3,3).b(3,2)$$

Figure 4.7 shows a schematic diagram that corresponds to clock cycle number 1, the same cycle is also shown in Fig. 4.8, in which elements a(1,1) and b(1,1) have entered cells numbers 14 and 15 respectively. From Fig. 4.7 it can also be seen that the element a(3,1) (green element) will enter into the array (at cell 06) at clock cycle 3 (see Fig. 4.7), while element b(1,2) (red element) will enter at clock cycle 2 (at cell 12) (see Fig. 4.9). After that, they will run through the array and will meet each other at cell 02 at clock cycle 5 (see Figure 4.12). This meeting generates the first partial result a(3,1).b(1,2). This can

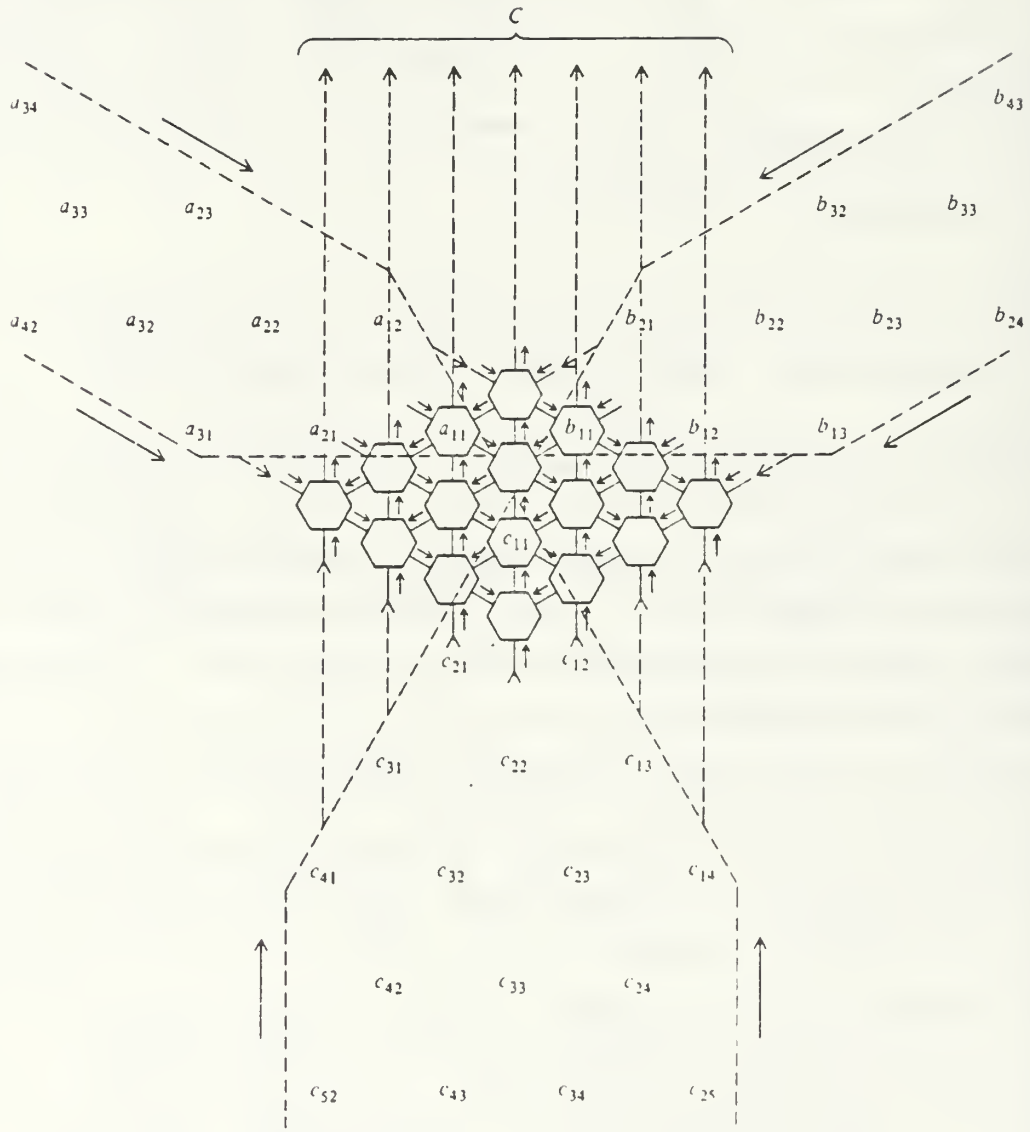



Figure 4.7 Systolic Array for Matrix-Matrix Multiplication

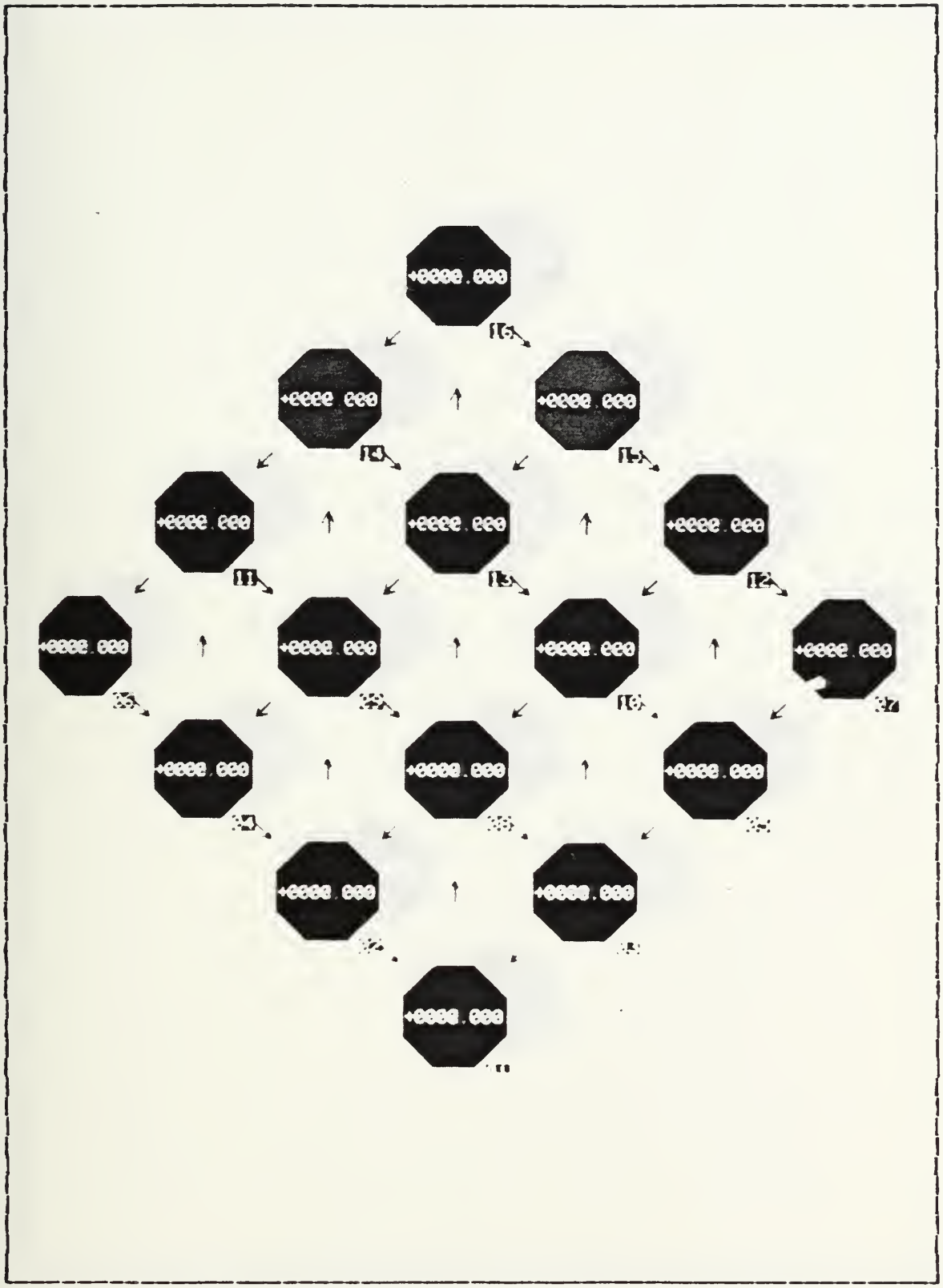


Figure 4.8 Matrix-Matrix Multiplication after Clock Cycle 1

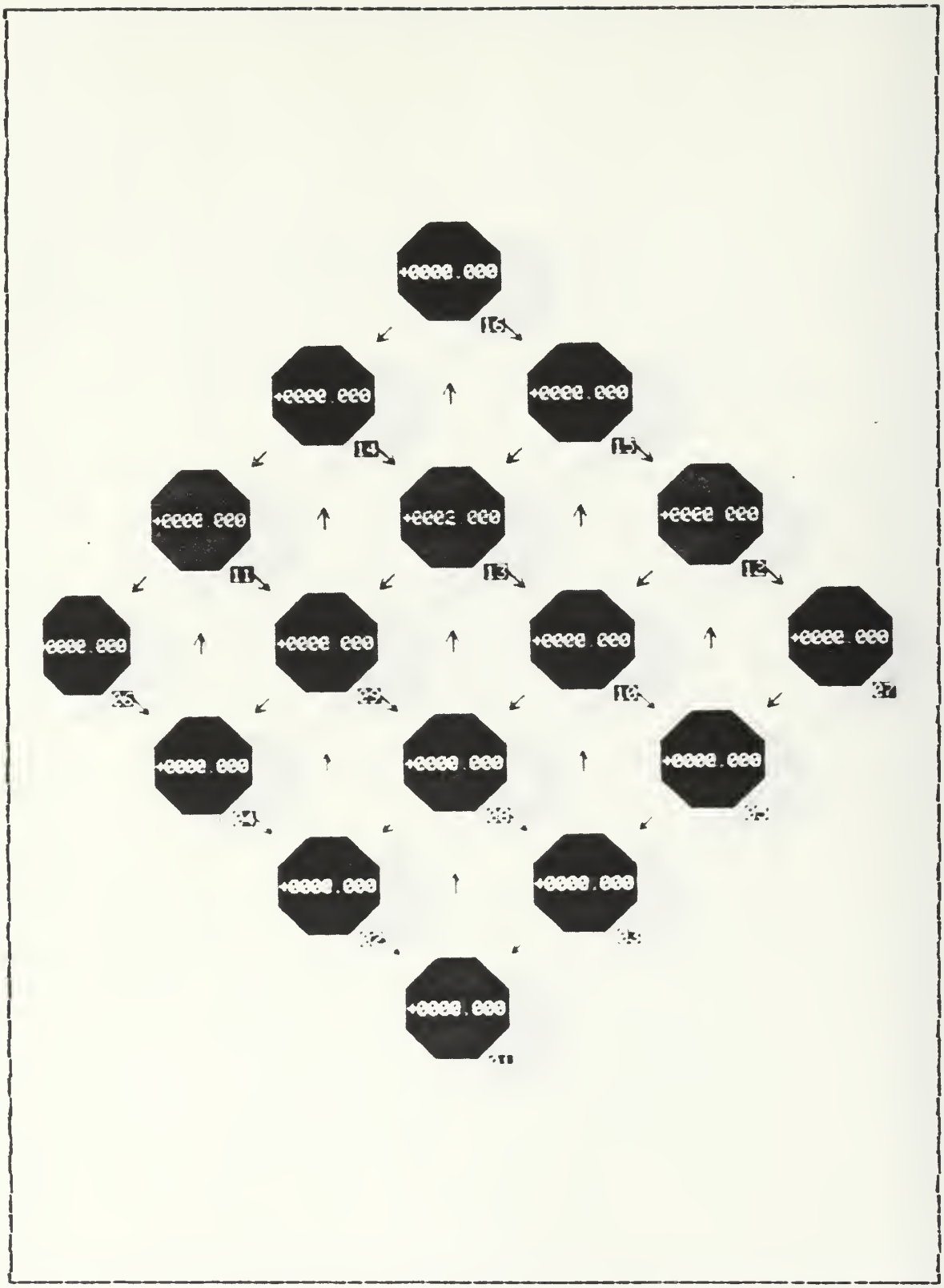


Figure 4.9 Matrix-Matrix Multiplication after Clock Cycle 2

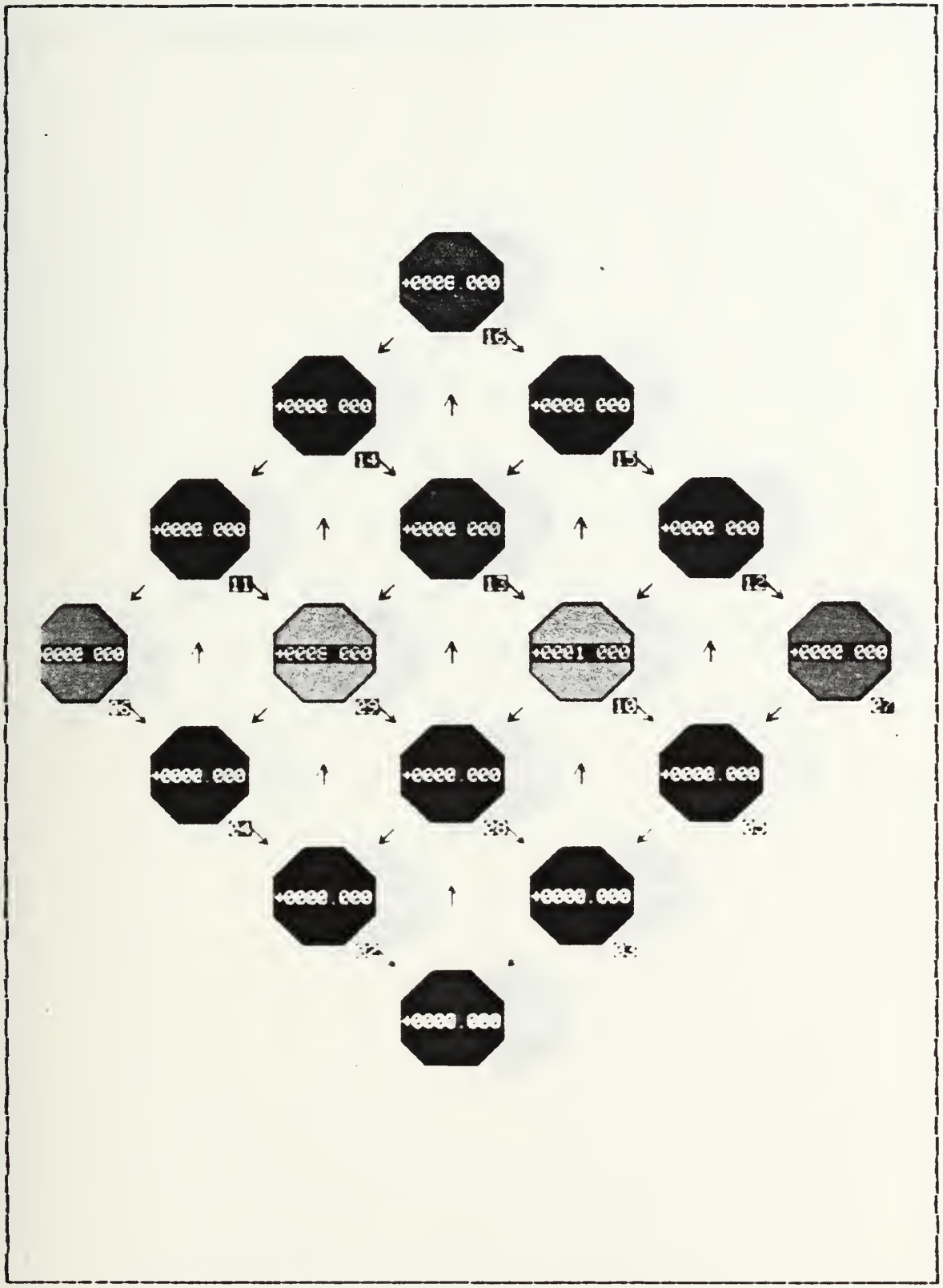


Figure 4.10 Matrix-Matrix Multiplication after Clock Cycle 3

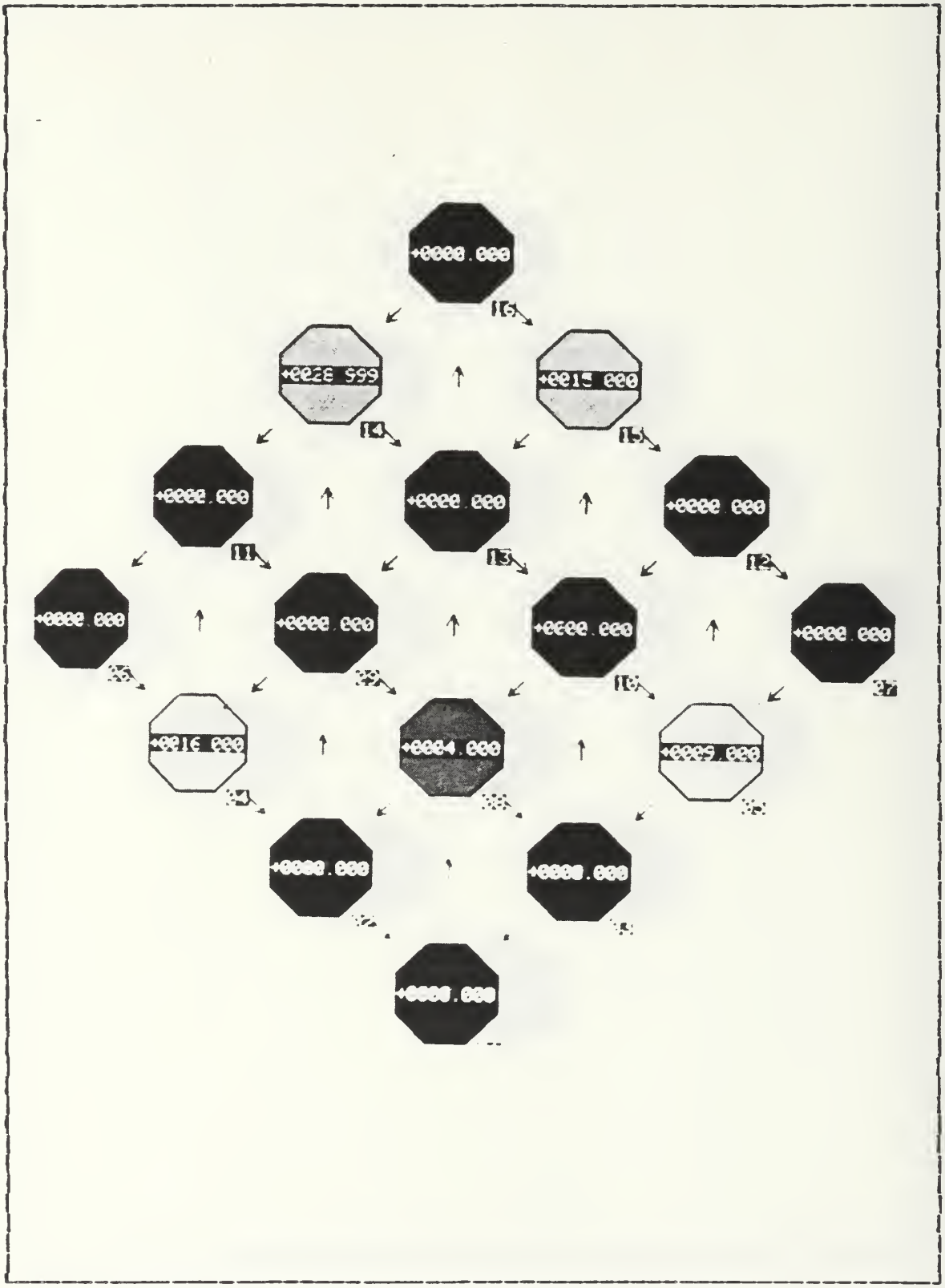


Figure 4.11 Matrix-Matrix Multiplication after Clock Cycle 4

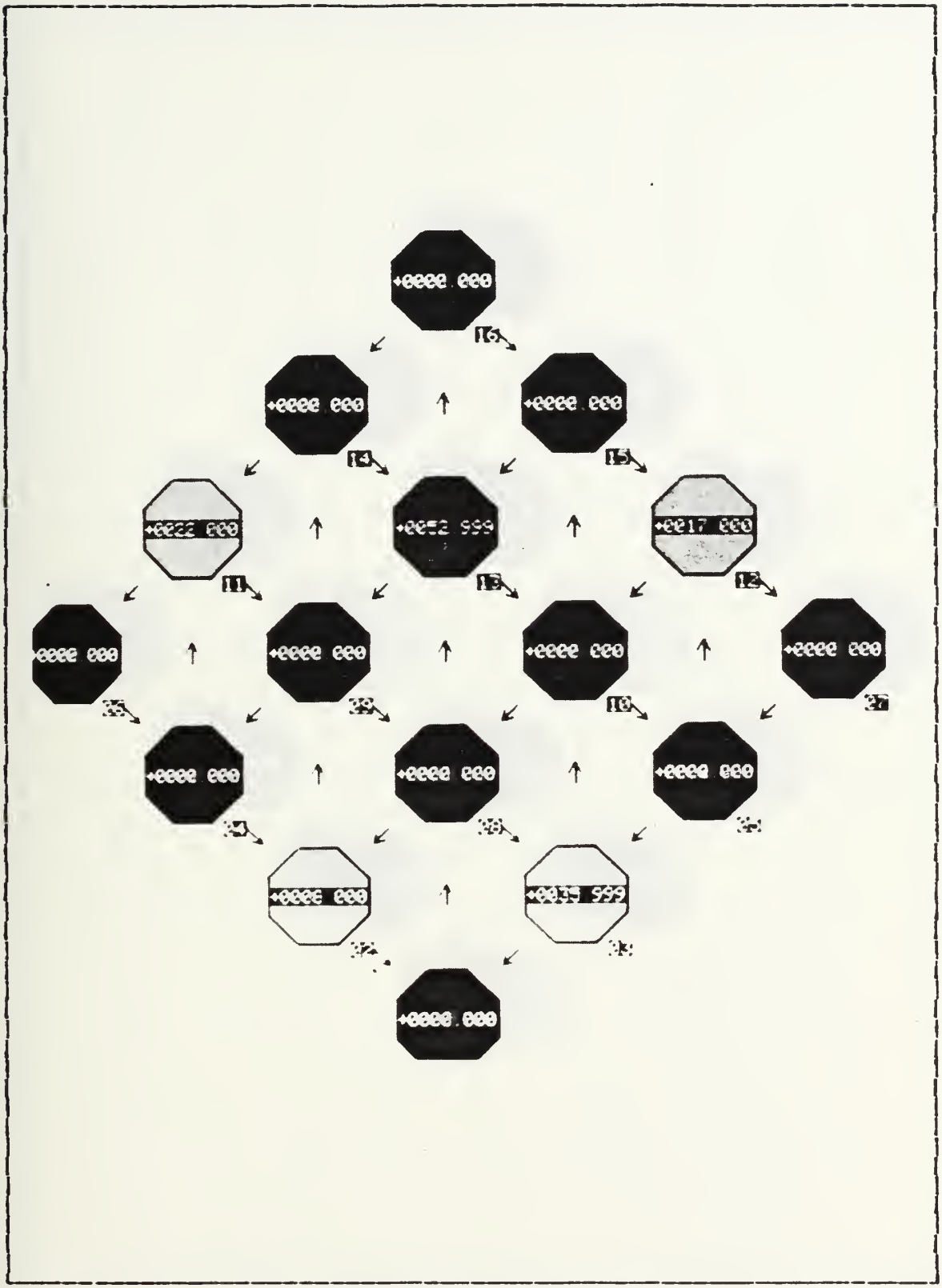
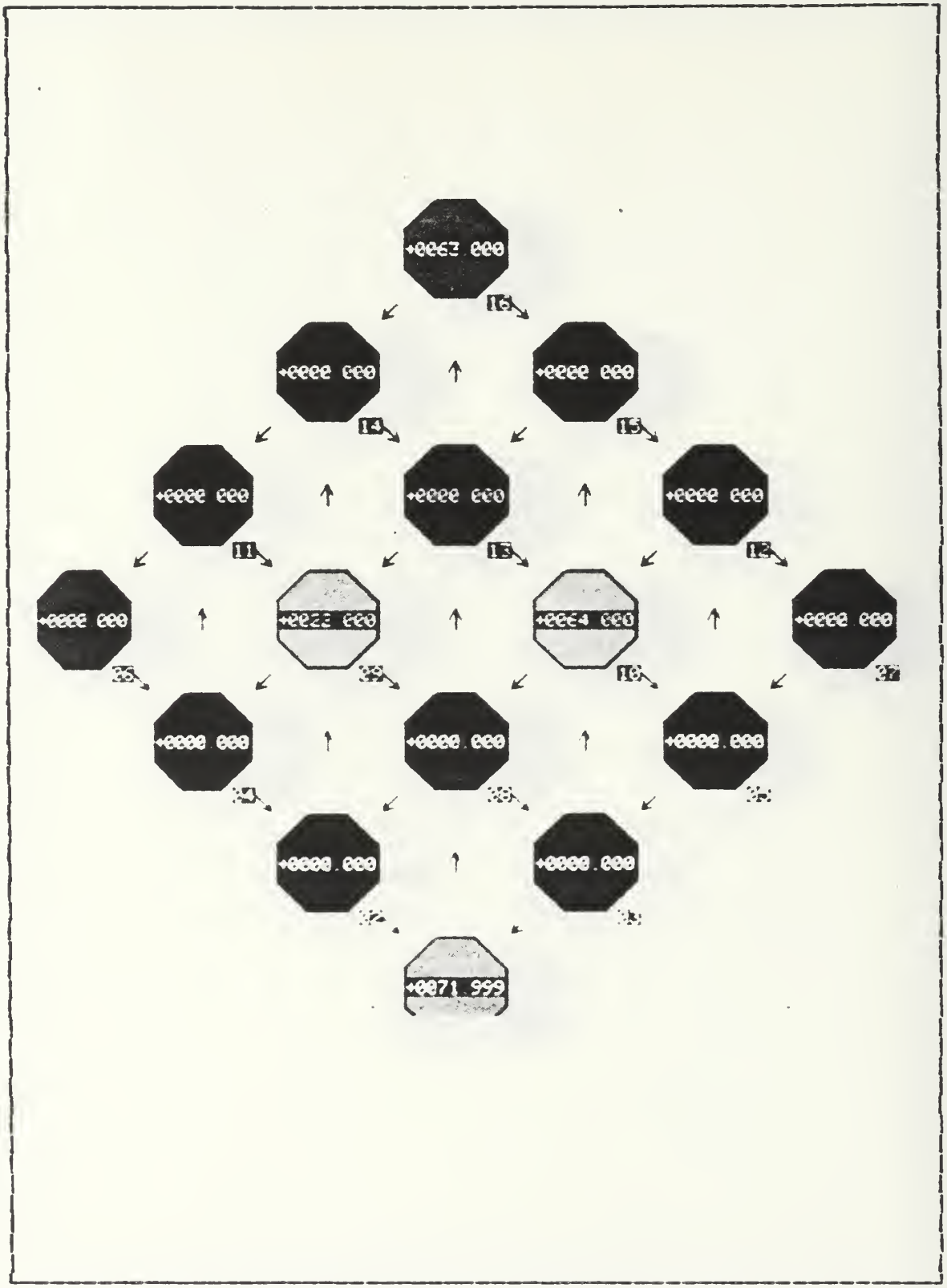


Figure 4.12 Matrix-Matrix Multiplication after Clock Cycle 5


+0000.000
+0000.000
+0000.000

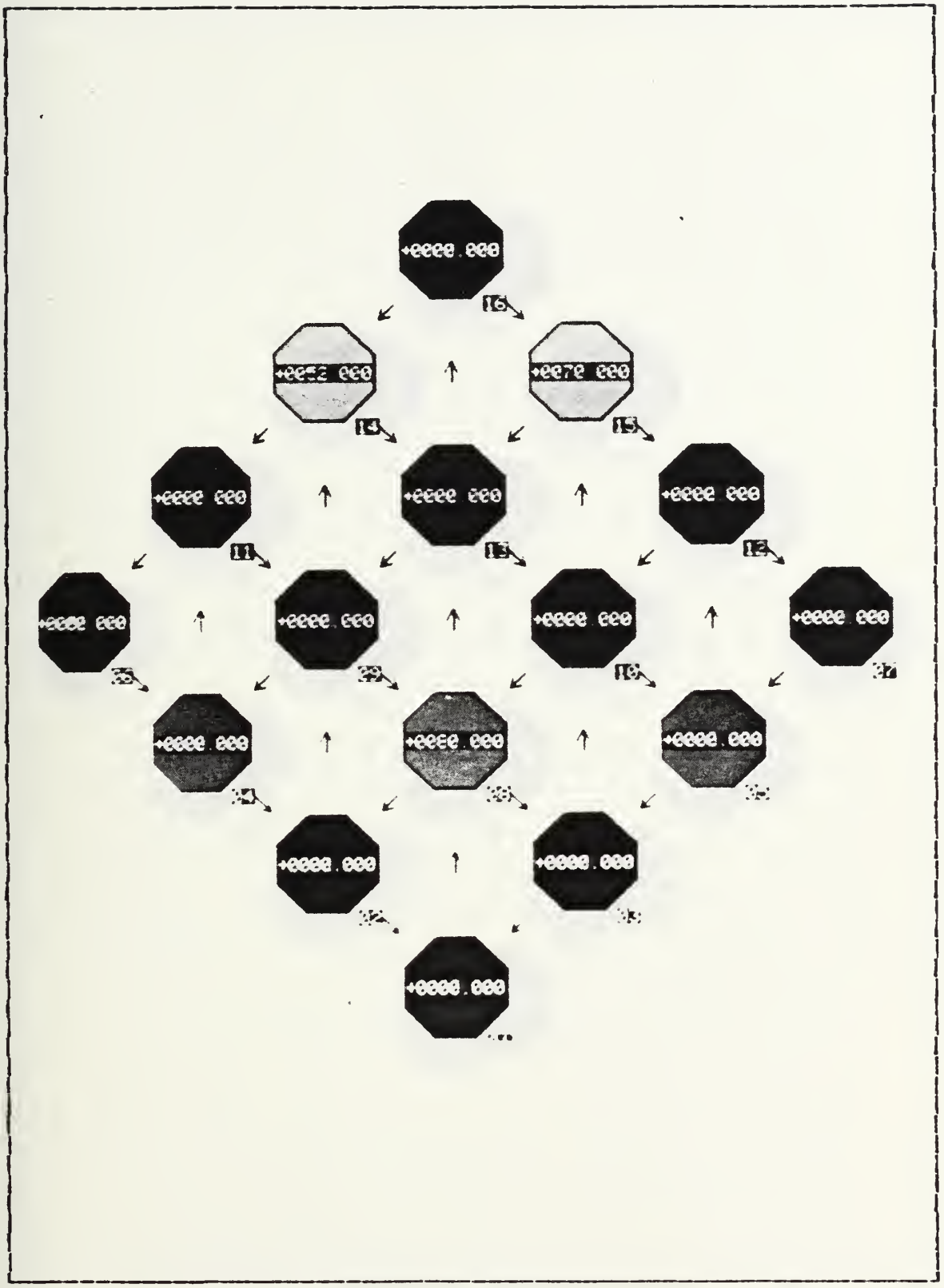


Figure 4.14 Matrix-Matrix Multiplication after Clock Cycle 7

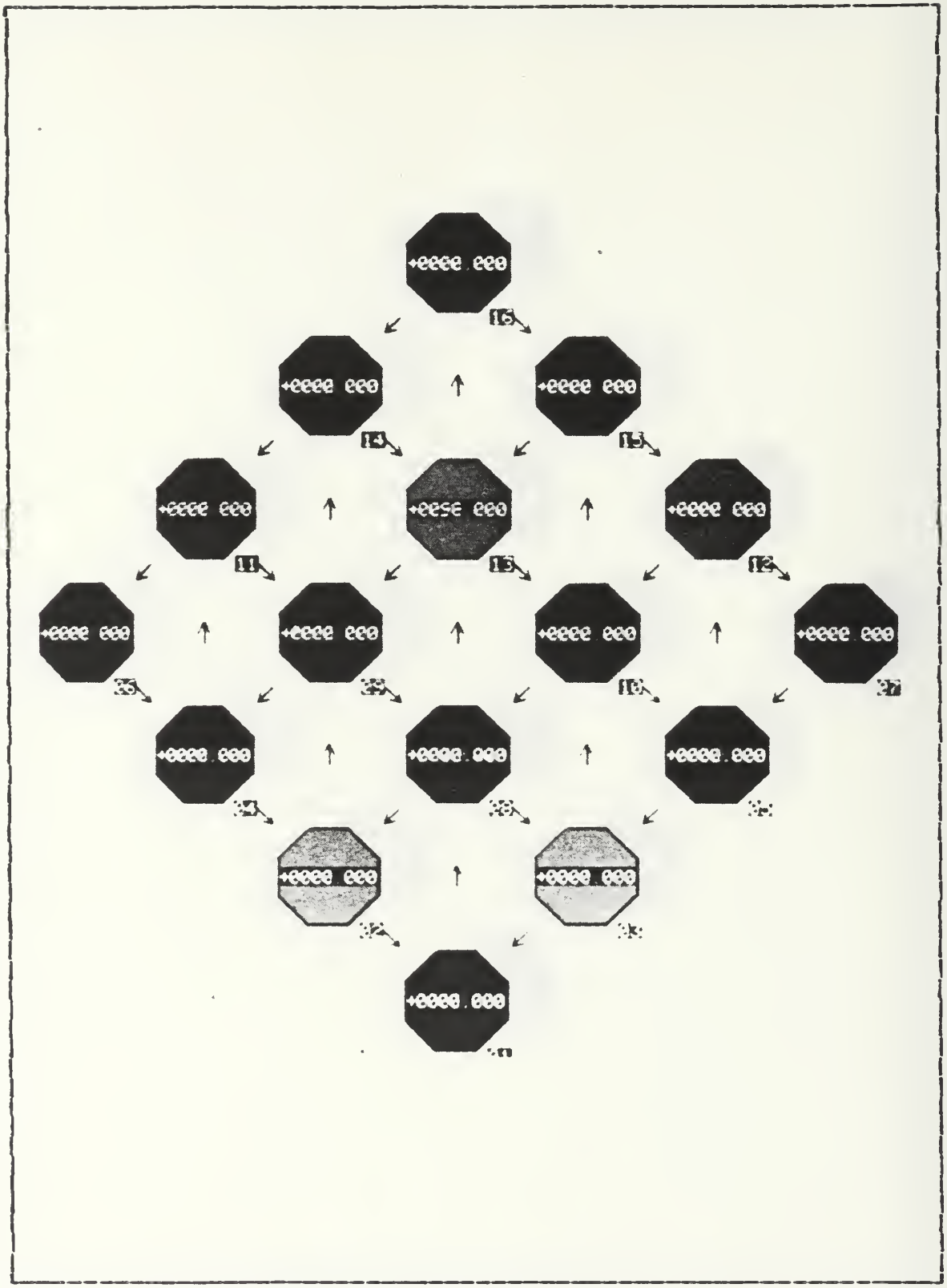


Figure 4.15 Matrix-Matrix Multiplication after Clock Cycle 8

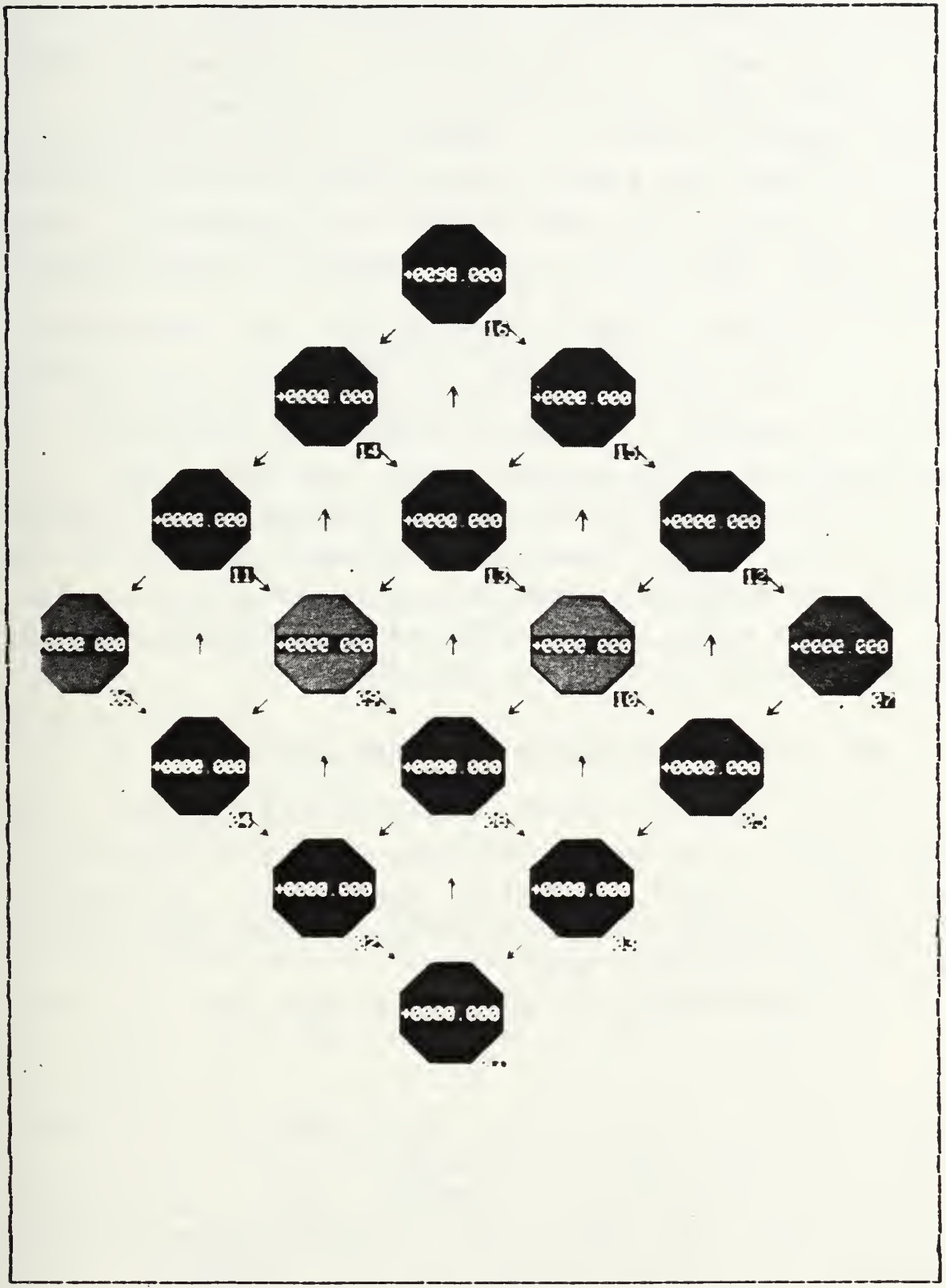



Figure 4.16 Matrix-Matrix Multiplication after Clock Cycle 9

be verified to be 8x1=8 by our numeric example and by the contents of cell 02 at clock cycle 5 . This partial result with yellow color (identified as c(3,2) for convenience) is pumped upwards to cell 09. Performing similar verification, it can be seen that elements a(3,2), b(2,2) and that partial result of c(3,2) will meet at cell 09 at clock cycle 6 (see Fig. 4.13). They will, then, generate a second partial result

$$a(3,1).b(1,2) + a(3,2).b(2,2)$$

which is 8x1+2x7=22 as seen in Fig. 4.13 at cell 09. By similar reasoning, the final result 8x1+2x7+3x10=52 is generated in cell 14 at clock cycle 7 (see Fig. 4.14).

It can be seen that matrix C is symmetrical with respect to the color coding. Symmetrical elements like C(3,2) and c(2,3) are computed in parallel and pumped up side by side. So, they are easily tracked with the help of colors, and the whole operation can be better visualized.

## C. MATRIX TRIANGULARIZATION BY GIVENS ROTATIONS

We will provide a brief review of this problem. The nomenclature to be used is the same as that in Chapter III. The reader is also referred to Appendix C, "A Numerical Example for Matrix Triangularization", which uses the same data as that described here.

The linear system that needs to be solved is

$$AX = B$$

where A is a nxp matrix, X is a column vector of p elements, and B, a column vector of n elements.

We will premultiply the matrix A by a transformation matrix Q such that the product matrix becomes an upper triangular matrix R. To keep the matrix equality it is necessary to premultiply vector B by matrix Q. This will

result in the product vector QB. This operation is shown below:

$$QAX = QB$$

and as R = QA, it beccmes

$$RX = QB$$

The Givens Rotations in this algorithm under simulation triangularize the matrix A and computes the vector QB concurrently. This is possible because

$$Q(A|B) = (QA)|(QB) = R|(QB)$$

where the operator "|" performs matrix concatenation. For clarity, if we define

$$A = \begin{bmatrix} 2 & 4 & 1 \\ 5 & 7 & 4 \\ 3 & 0 & 1 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 12 \\ 3 \\ 8 \end{bmatrix}$$

then A|B becomes

$$A|B = \begin{bmatrix} 2 & 4 & 1 & 12 \\ 5 & 7 & 4 & 3 \\ 3 & 0 & 1 & 8 \end{bmatrix}$$

As we have shown in Chapter III, the simulation uses the cell structure seen in Fig. 3.4 and the test data seen in Figure 3.5 (the same as that used in the numerical example cf Appendix C). The picture that appears on the screen at the start of the simulation is shown in Fig. 4.17 which the reader should compare with Fig. 3.4 in Chapter III. Fig. 4.17 shows the identification of each cell at its lower right corner. Cells numbered 13 to 10 represent the data wavefront to be pumped into the systolic array. They are not part cf the array, but only buffer cells to allow presentation of the data to be pumped into the array on the screen.

The final elements of vector QB are calculated in cells 04, 02, and 01. The final elements of matrix A are calculated in cells numbered 09, 08, 06, 07, 05 and 03.

As shown in Appendix C, the values for AX = B are as follows and A|B is as shown above.

$$\begin{bmatrix} 2 & 4 & 1 \\ 5 & 7 & 4 \\ 3 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x(1) \\ x(2) \\ x(3) \end{bmatrix} = \begin{bmatrix} 12 \\ 3 \\ 8 \end{bmatrix}$$

The test data are pumped into the array as wavefronts that can be seen in cells 13 to 10 just before being entered (the reader can refer to Fig. 3.5 to the several wavefronts, each one corresponding to a different color). There is a time shift among the elements of the same row for the matrix under triangularization (compound matrix A|B). This displacement is needed to establish the correct timing for the systolic operation. In this simulation, we decided to assign a different color for each row of A|B (as shown in Fig. 3.5) and to keep track of the flow of the colored elements through the array. This flow can help us to understand, in a sequence of space-time frames, how the whole array behaves. The effect of the different clock for subsequent cycles can be seen from Fig. 4.18 to 4.26 . In Fig. 4.18, element a(1,1) of matrix A enters into cell 09. We can see, in cell 13 element a(2,1), and in cell 12 element a(1,2). They are ready to be clocked in at the next clock cycle. This happens as seen in Fig. 4.19, when cells 09 and 08 display the stored result of the computation performed. Notice that cells 09, 07, and 03 actuate like a reflector for the data wavefront that is going downwards. They rotate the data flow direction by 90 degrees counterclockwise. As a result of the interaction between the wavefronts that go towards the right and that going downwards, there is a resulting wavefront that flows towards the lower

right. This is the important observation because it shows how the effect of the input data spreads over the array (this is why we decided to associate this wavefront with identical colors). The wavefront flowing towards the right carries information about the angle that the row elements must be rotated. This information must arrive at each cell just in phase with information of the matrix element being rotated, which is being transferred to each cell by the downward wavefront. At each clock cycle a new rotation angle is calculated at the inclined boundary cells (Givens External Cells, namely 09, 07 and 03) and transmitted to the rightest cells at the same row (that is from cell 09 to cells 08, 06 and 04 in the upper row, from cell 07 to cells 05 and 02 in the second row and from cell 03 to cell 01 in the lower row). Partial results are generated at each systolic array element at each clock cycle. At the moment when the downward wavefront start feeding zeros into a systolic array element, it freezes its content and do not modify it any more. We have selected the black color to indicate that the input data are bringing no information. The black wavefront flowing through the systolic array acts as a freezing wavefront. Figs. 4.19 up to 4.26 show the effect of the remaining clock cycles. In Fig. 4.26 we have the final result of the triangularization frozen into the cells. The upper triangular matrix R and the vector QB have been computed. It can be checked out against those values shown in Appendix "A Numerical Example for Matrix Triangularization". The computed results are ready to be used in the Back Substitution to solve the system equations. In order to transfer the data from the cells to the hardware that performs the Back Substitution operation, a special connection which is not shown here becomes necessary. But, it is not required here for the understanding of the Givens Rotations process.

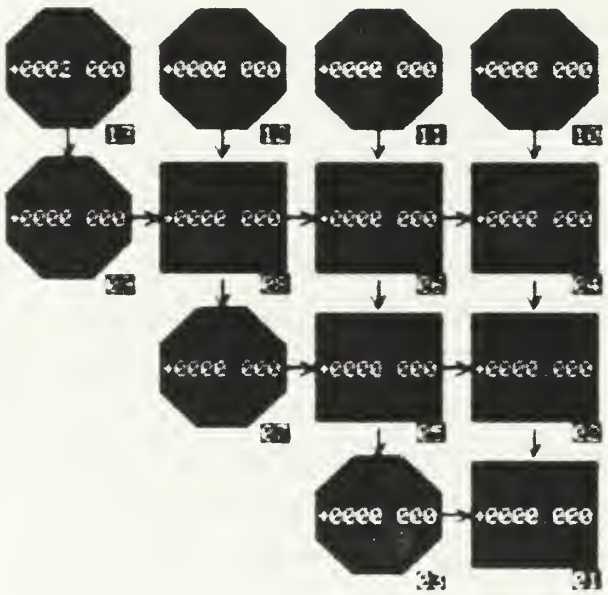


Figure 4.17 Matrix Triangularization Array Initialization

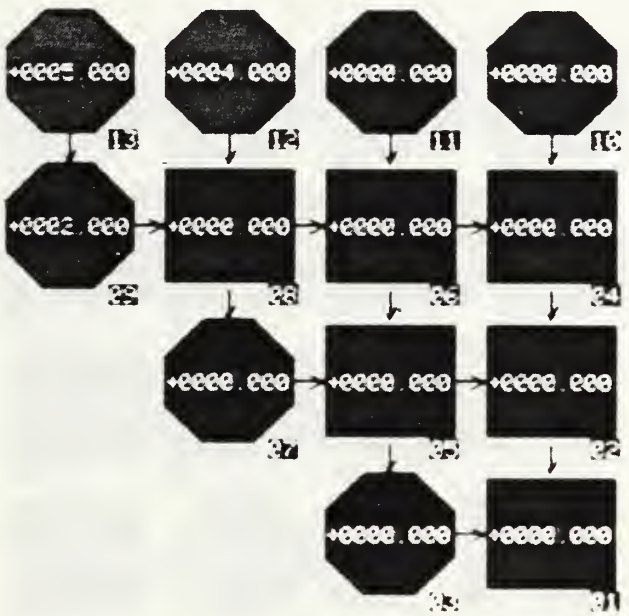


Figure 4.18 Matrix Triangularization Array after Clock Cycle 1

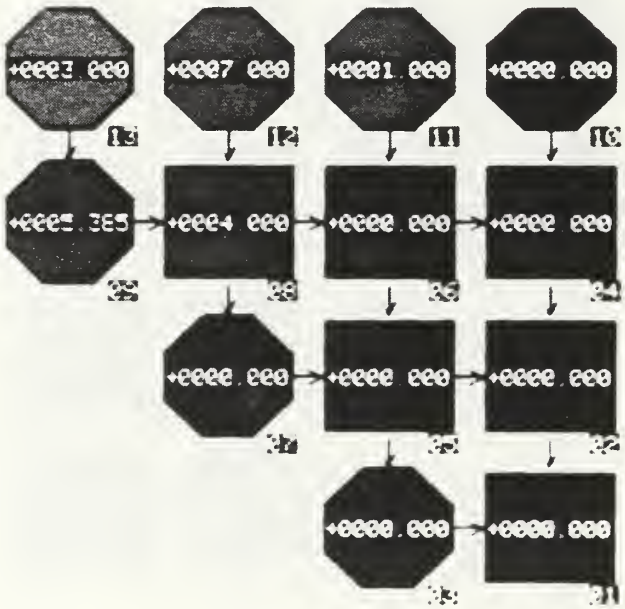


Figure 4.19 Matrix Triangularization Array after Clock Cycle 2

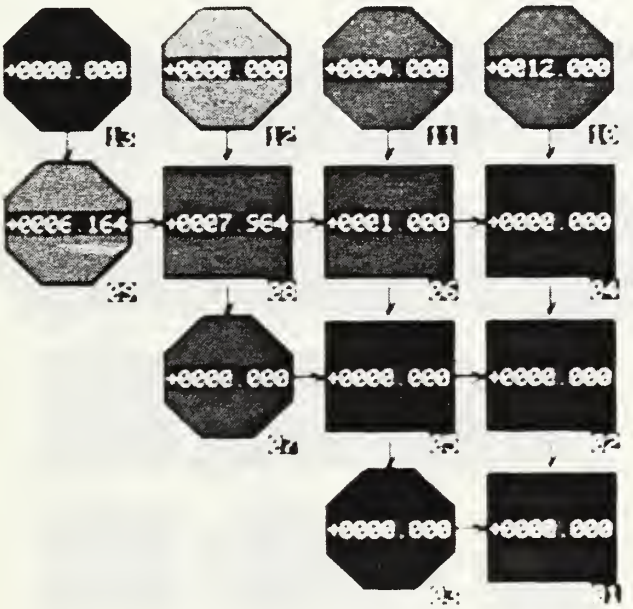



Figure 4.20 Matrix Triangularization Array after Clock Cycle 3

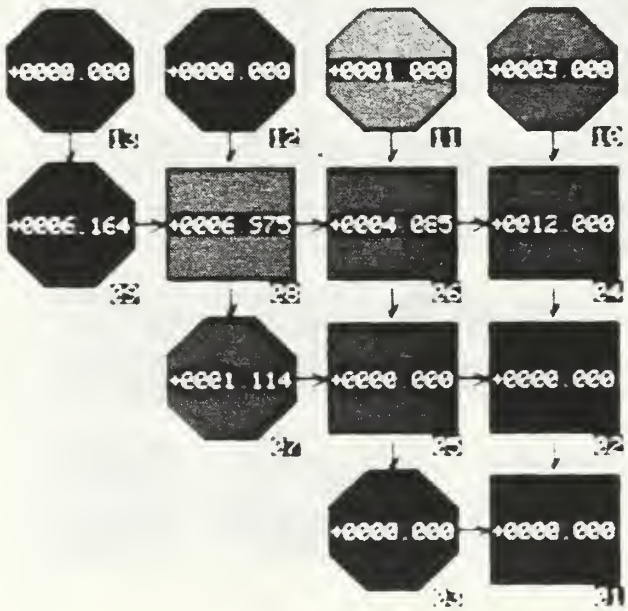



Figure 4.21 Matrix Triangularization Array after Clock Cycle 4

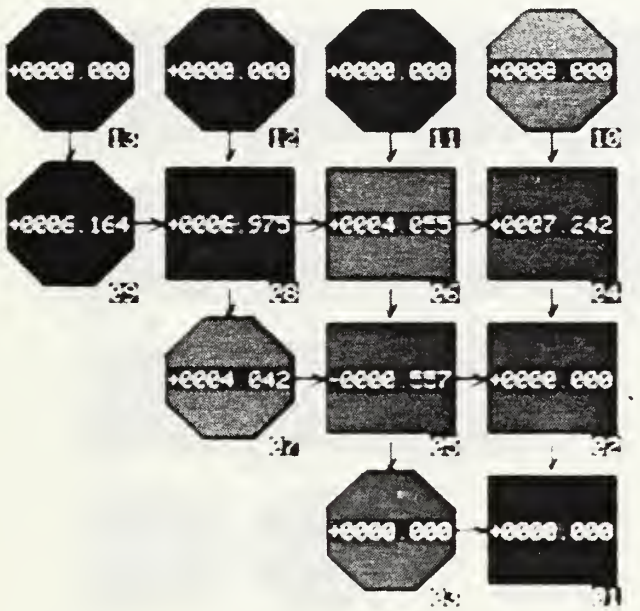



Figure 4.22 Matrix Triangularization Array after Clock Cycle 5

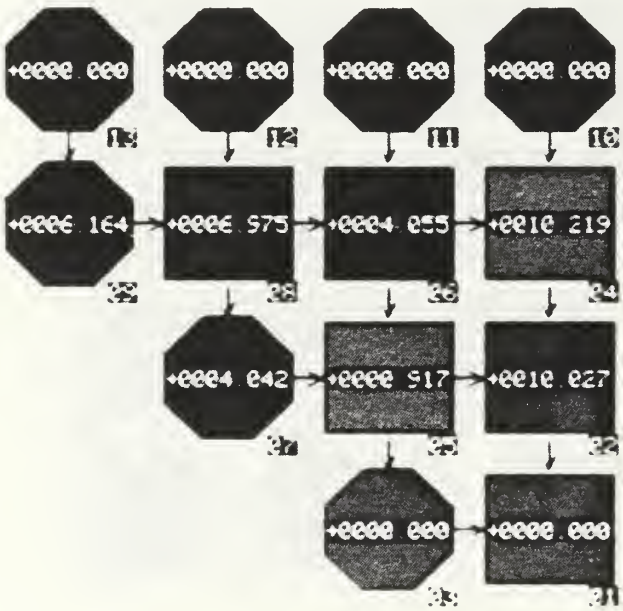



Figure 4.23 Matrix Triangularization Array after Clock Cycle 6

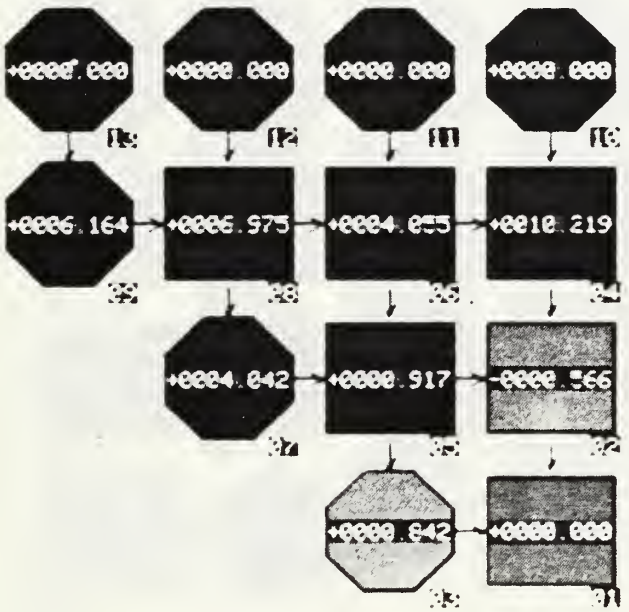



Figure 4.24 Matrix Triangularization Array after Clock Cycle 7

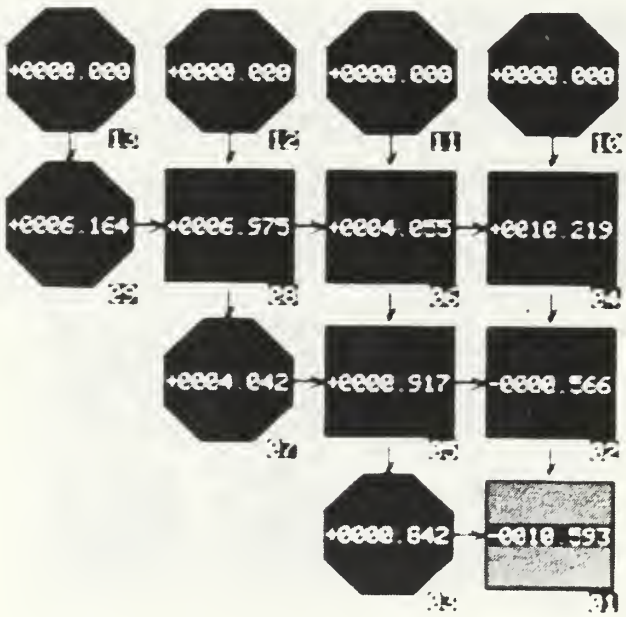



Figure 4.25 Matrix Triangularization Array after Clock Cycle 8

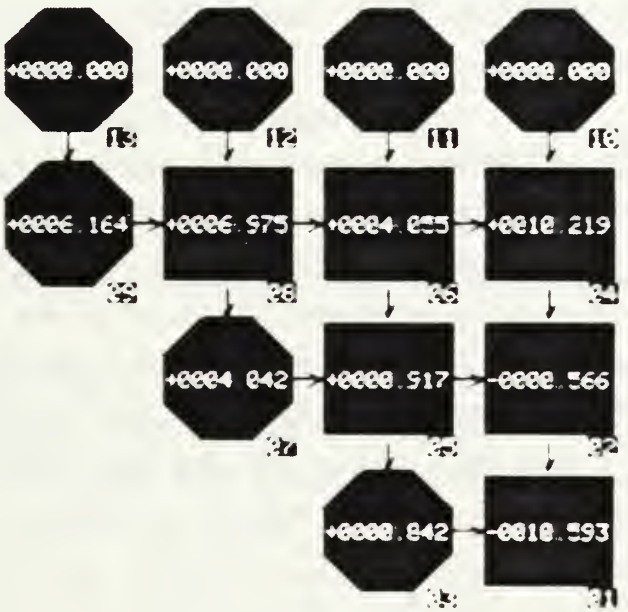



Figure 4.26 Matrix Triangularization Array after Clock Cycle 9

We will now present a different kind of perspective to the reader on how the algorithm performs. The geometrical interpretation of Givens Rotations is extremely helpful to provide better insight. A matrix can be considered as representing a set of vectors (as many as the number of columns) in a n-dimensional space (n equals the number of rows). If the matrix has only three rows, their columns are vectors of a three dimensional space, and the elements of the first row are the components over the x-axis of the column vectors. The rotation operation does not rotate the vectors. The reference frame (coordinate axes) is the one that is rotated, and, as a consequence, the description of the vectors in terms of their components with respect to the new reference becomes different. When a matrix is rotated to become an upper triangular, the first column is transformed in such a way that only element (1,1) will be nonzero. This means that the first column vector is positioned over the x-axis in the new reference frame. As the vector is the same as before, the numerical value of the new element (1,1) must be equal to the length of the vector. Element (1,2) of the rotated matrix, for example, is the x-component of the second column vector in the new reference frame. Since the relative position of both vectors is unaltered, the new value of that element must be equal to the projection of the second column vector over the first column vector, since this last one is now along the new x-axis. The geometrical interpretation can be extended to all elements of the array.

We will track the build up of the numerical values of cells 09 (that stores element (1,1)) and 08 (that stores element (1,2)). This allows us to study the operation of both types of cells used in this algorithm (cell 09 is an outer cell and cell 08 is an inner cell). It will also provide a comparison between the geometric interpretation presented above and the algebraic values shown in graphics.

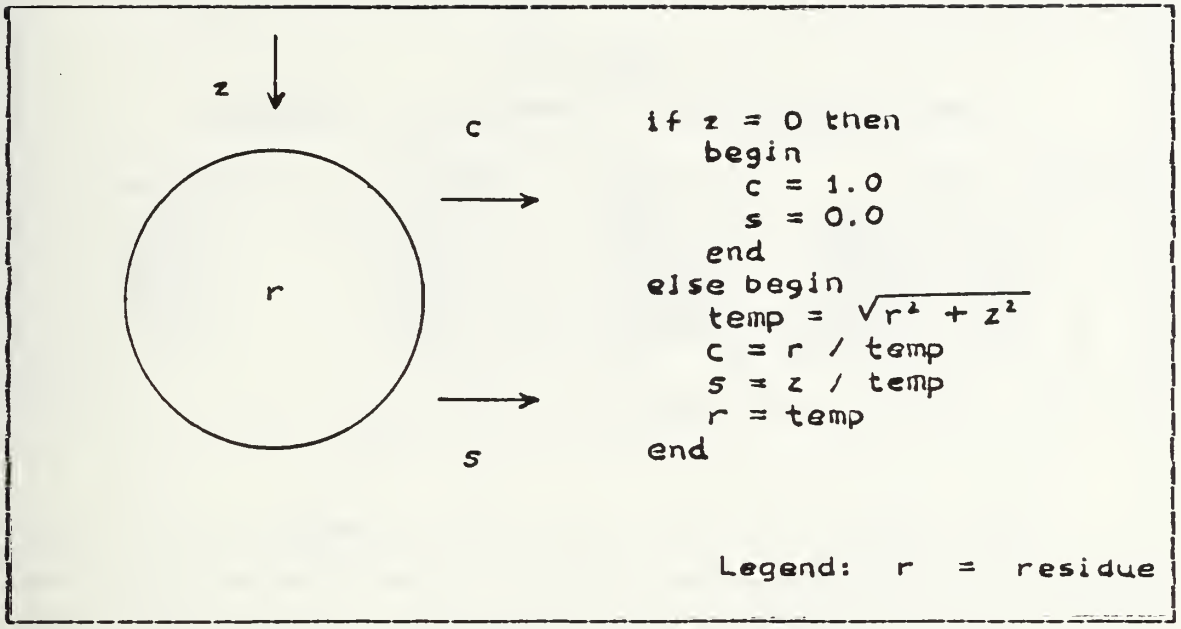



Figure 4.27 Givens External Processor Cell

We will start with cell 09. From Fig. 4.27 we see that this cell receives the elements of the "nonrotated" first column vector, computes the rotation angle (calculating its outputs cosine c and its sine s), and stores the "rotated" x-component r of the first vector. The other components are obvicusly zeros and ccrrespond to the other elements of the first cclumn (cells cf the lcwer array that are not shown because their ccntents are always zeros). The rotation angle information is passed as output to cell 08 to be used to "rotate" the second vector. Table I shows how the parameters r, c, and s of this cell respond to the external inputs. These values can be verified with the help of the cell algcrithm presented in Fig. 4.27 . The descriptiон cf the cperation can be followed through the sequence of Figs. 4.28 to 4.30.

**TABLE I**

**Time Description of Outer Cell Operation**

| Cycle | Input z | Output c | Output s | Residue r |
|---|---|---|---|---|
| 0 | 0.0 (black) | 1.000 | 0.000 | 0.000 |
| 1 | 2.0 (blue) | 0.000 | 1.000 | 2.000 |
| 2 | 5.0 (red) | 0.371 | 0.928 | 5.385 |
| 3 | 3.0 (green) | 0.874 | 0.487 | 6.164 |
| 4 | 0.0 (black) | 1.000 | 0.000 | 6.164 |

At clock cycle 0 no input has been received and so there is nothing to rotate. The rotation angle is zero (c=1.0 and s=0.0). At cycle 1, the first element is received. The first element is the old x-component of the first column vector. As it is over the x-axis, the reference does not need to be rotated now. However, when this occurs, because of the way the algorithm is implemented (we will see the reason when analysing cell 08), an angle 90 degrees is informed to cell 08 (c=0.0 and s=1.0) although the value r stored in cell 09 is 2.0, equal to the input. At clock cycle 2, the y-component of the old first column vector is entered. Now a rotation is necessary to keep the x-axis of the reference frame over the first column vector. The rotation angle is computed (c=0.371 and s=0.928) and the reference frame is rotated (about the z-axis). The new x-component (stored in r) is the square root of the summation of the squares of the old x and y-components, that is 5.385 (corresponding to the projection of the first column vector over the x-y plane). At cycle 3, the z-component is entered. As the last rotation resulted in a vector over the x-axis, the combination of this with the just entered z-component will result in a vector in the x-z plane deviated from the x-axis because of the newly arrived z-component. Again a rotation is necessary

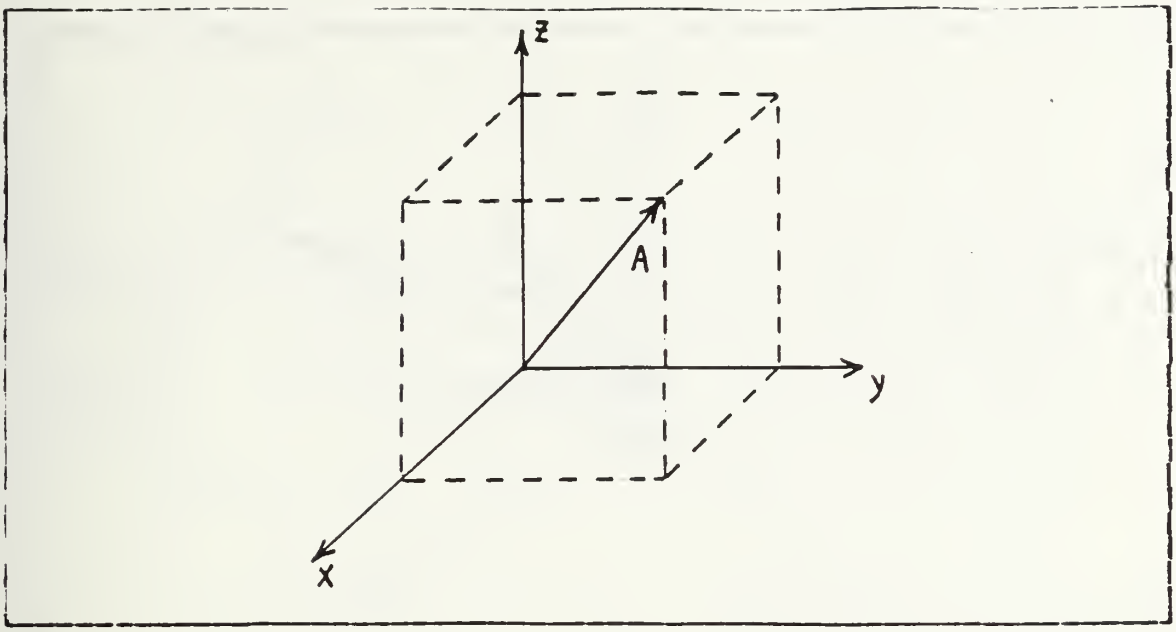



Figure 4.28 Reference Frame before Rotations

to keep the x-axis over the first column vector. The new x-component is 6.164. At clock cycle 4 a black datum is received (a zero value) and that freezes the contents of cell 09 with its final value. No more rotations are required and the output of the cell informs rotation angle zero (c=1.0 and s=0.0).

Now we will follow the build up of the contents of cell 08 (the right side neighbor cell of cell 09). Cell 08 stores element (1,2), the x-component of the second column vector. As done before, we present table II with inputs and values stored in this cell at each clock cycle. In this case we will disregard the numerical output of this cell because we will concentrate on the build up of the residue r in this cell. From Fig. 4.31, this cell receives as input the rotation angle (inputs c and s), by which the reference frame has changed, to be used to compute the new x-component of the second column vector (to be stored at r). The other input refers to the elements of the "nonrotated" second

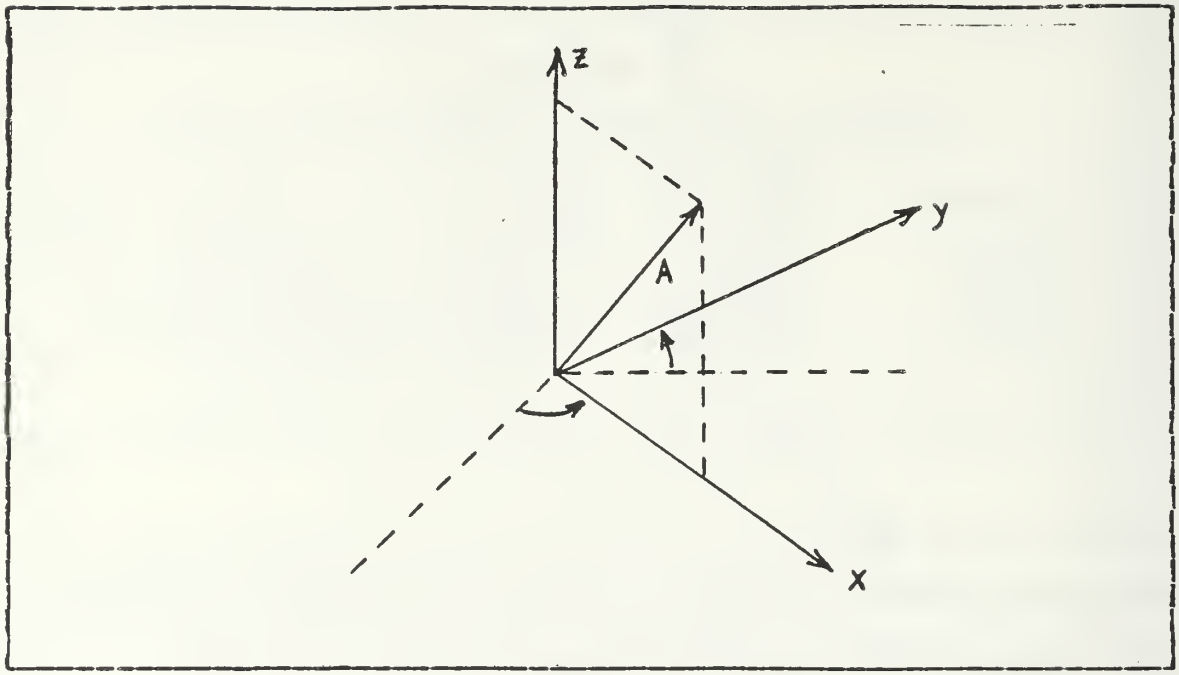



Figure 4.29 Reference Frame after First Rotation

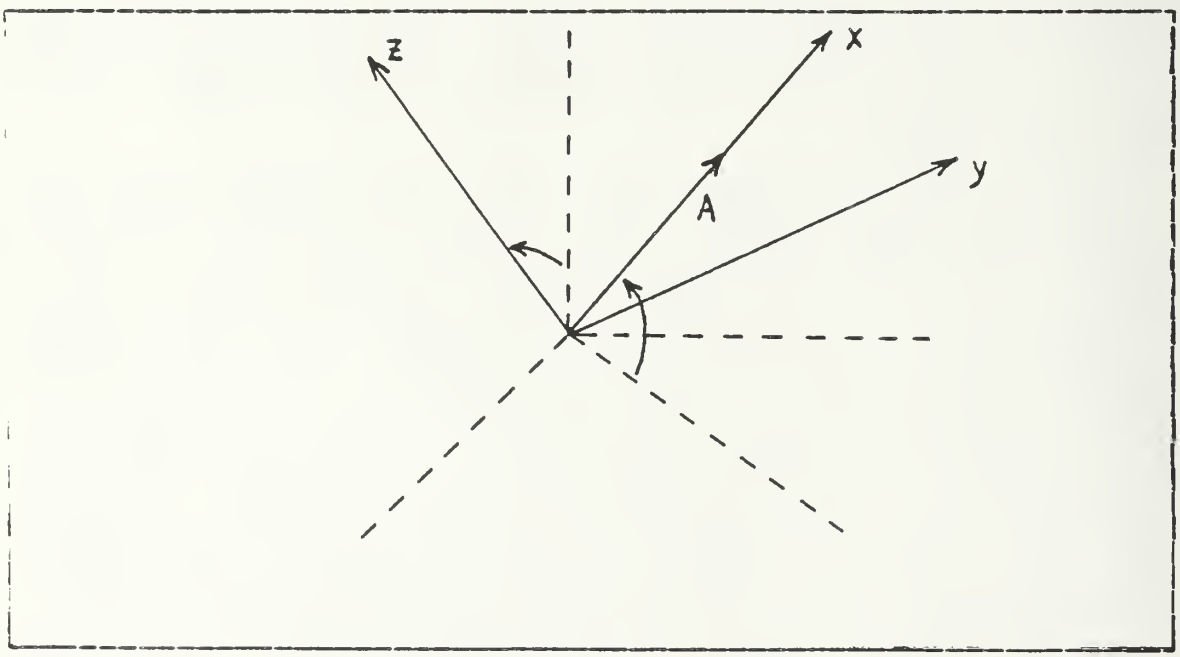



Figure 4.30 Reference Frame after Second Rotation

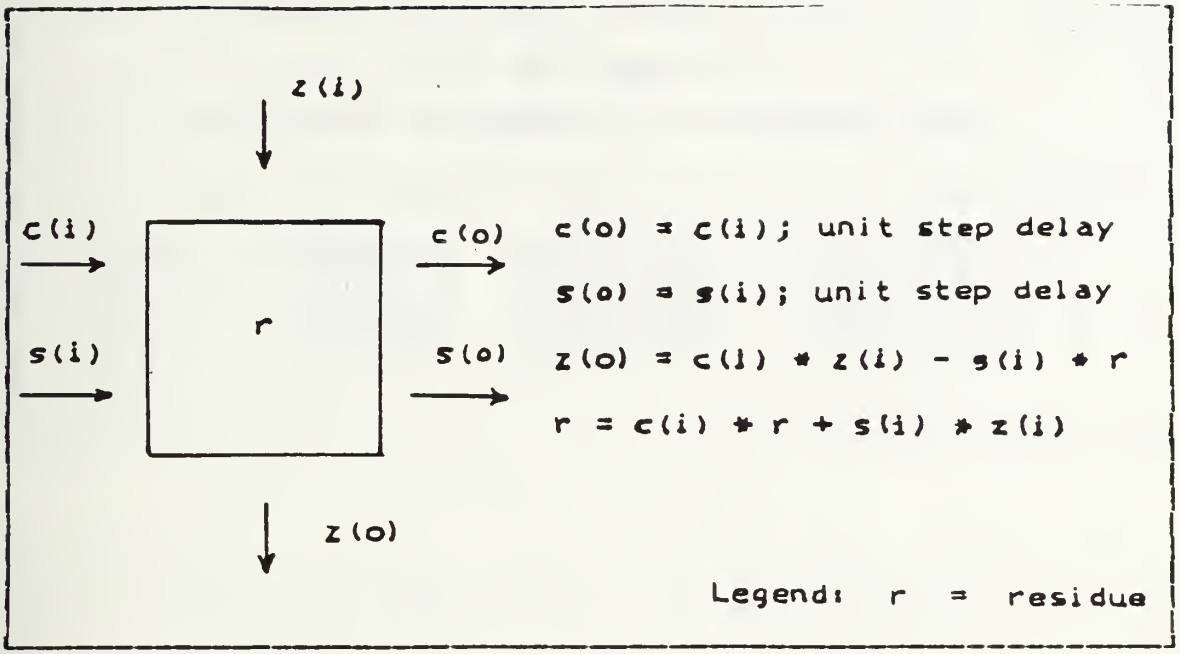



Figure 4.31 Givens Internal Processor Cell

column vector (input z). The outputs of this cell are the rotation angle (c and s are passed as output to the right neighbor cell 05) and rotation information to the neighbor cell immediately below cell 08 to "rotate" the other components of the column vector as a result of the reference frame rotation.

The operation of this cell is more difficult to visualize. Figure 4.32 will be used for the explanation. It only shows the x-y plane. Suppose the second column vector is A, as seen in that Figure. The old reference frame is x1-y1. The components of A with respect to that frame are ax and ay. In our numerical example, ax=4.0 is the first input z(i) to the cell. The second input z(i) to the cell, on the following cycle, is ay=7.0. Suppose the reference frame is rotated about the z-axis to position x2-y2 to keep the x-axis along the first column vector, which is shown as B.

**TABLE II**

**Time Description of Inner Cell Operation**

| Cycle | Input c | Input s | Input z | Residue r |
|---|---|---|---|---|
| 0 | 1.000 (black) | 0.000 (black) | 0.0 (black) | 0.000 |
| 1 | 1.000 (black) | 0.000 (black) | 0.0 (black) | 0.000 |
| 2 | 0.000 (blue) | 1.000 (blue) | 4.0 (blue) | 4.000 |
| 3 | 0.371 (red) | 0.928 (red) | 7.0 (red) | 7.985 |
| 4 | 0.874 (green) | 0.487 (green) | 0.0 (green) | 6.976 |
| 5 | 1.000 (black) | 0.000 (black) | 0.0 (black) | 6.976 |

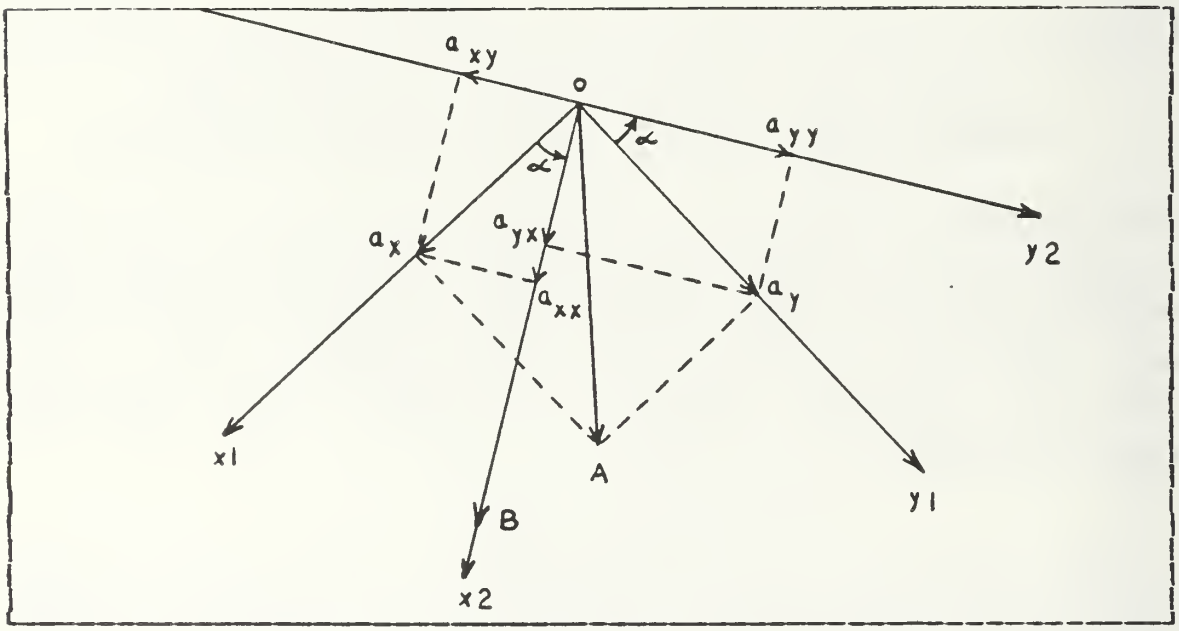



**Figure 4.32 Geometric Interpretation of a Rotation**

Instead of evaluating the components of A in this new frame, let us study the effect over its components ax and ay. In this new frame, ax will be decomposed into components axx (over the new x-axis) and axy (over the new y-axis). Similarly, ay will be decomposed into components ayx (over

the new x-axis) and ayy (over the new y-axis). The component of A over the new x-axis is the summation of axx and ayx. This will be the new element (1,2). As seen in Fig. 4.32,

axx = ax . cos(alfa) and ayx = ay . sin(alfa)

and as element (1,2) is

r = axx + ayx

it results

r = ax . cos(alfa) + ay. sin(alfa)

As ax was the previous r and ay is the newly arrived input z(i), in computer algorithmic language we have

r = c(i) * r + s(i) * z(i)

as in Fig. 4.31 . Substituting the values of clock cycle 2 into this equation, we find

r = 0.0 * 0.0 + 1.0 * 4.0

r = 4.0

and this can be checked against the above table. At clock cycle 3, the values give us

r = 0.371 * 4.0 + 0.928 * 7.0

r = 7.985

that also checks against the above table. Doing the same for clock cycle 4, we get r=6.976. This gives the final value of the x-component of the second column vector. The rotation, however, will affect all other components. Let us describe how this occurs. The component of A over the new y-axis is

z(o) = ayy - axy

z(o) = ay . cos(alfa) - ax . sin(alfa)

and in algorithmic language,

$$z(o) = c(i) * z(i) - s(i) * r$$

This infcrmation z(c) is passed to the cell immediately below tc generate the new y-component of the second column vector.

A further point to be noticed is that if the rotation angle input to an inner cell is zero (c=1.0 and s=0.0), the cell will not change the stored value at r. This is why the rotation angle is informed as 90 degrees, as mentioned previously, when the first element is received at cell 09. This triggers cell 08 to receive and store the z input at the follcwing clock cycle. At the end of clock cycle 02, the z=4.0 input is stored. No rotation is performed, so r=4.0. The follcwing clock cycles impose the modificatien cf the contents of this cell because of the changes in the reference frame. As soon as the input z freezes (at clock cycle 5), the residue r cf the cell also freezes. It can be cbserved, if the outputs of cell 09 and its inputs tc cell 08 are ccmpared, the transmmission delay of one clock cycle from a cell tc another. It can also be noticed that the inputs tc cell 08 always have the same color. This simulation was purposely designed this way to show the need for sinchronization between the wavefronts that propagate through the array.

Certainly we cannct present all this complexity with the simulator. The algorithm of Givens Rotations is the most complex that we have traced in our survey. However, the simulator has been a fundamental tool to perform a numerical study and to verify the correctness of the algorithm.

## D. BACK SUBSTITUTION

We will use the results of the previous section as input data for the simulation described in this section. This way, with these two sections, we can go through the complete problem described in the Appendix "A Numerical Example for Matrix Triangularization". That is, to search for the vector X in matrix equation AX=B. The process of Back Substitution is described in [Ref. 7: page 24].

One of the cells used in this algorithm is shown in Figure 4.33 . The above reference presents the cell shown in Fig. 4.33 as a circular shaped cell named Back Substitution Cell. Since SYSGRAS would take too much time to draw circular shapes, an octogonal shape is used instead (see cell number 05 in Figure 4.37).

The other type of cell used in this algorithm is the same Inner Product Step Processor that was presented in Fig. 4.6 to do Matrix-Matrix Multiplication. This is an interesting aspect of systolic arrays: many algorithms that appear in the literature are implemented with the same types of cells. However, the geometry of the array, in this algorithm, requires the cell to be square shaped. Although the cell is functionally the same as shown in Fig. 4.6, we present it again in Figure 4.34 to match the way Fig. 4.35 represents it as part of the structure (see cells 04 and 02).

The mathematical background for solving a triangular linear system involving a lower triangular array has been presented by Kung in [Ref. 1: page 19]. We will adopt here his explanation.

Suppose the system equation to be solved is

$$RX = D$$

where R = (r(i,j)) is a nonsingular nxn lower triangular

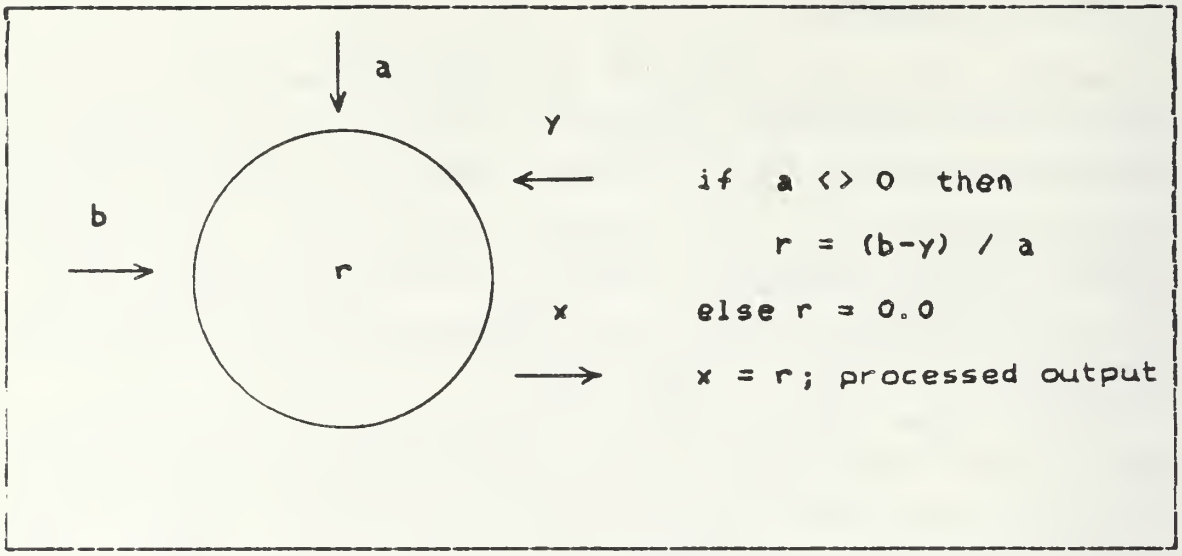



Figure 4.33 Back Substitution Main Cell

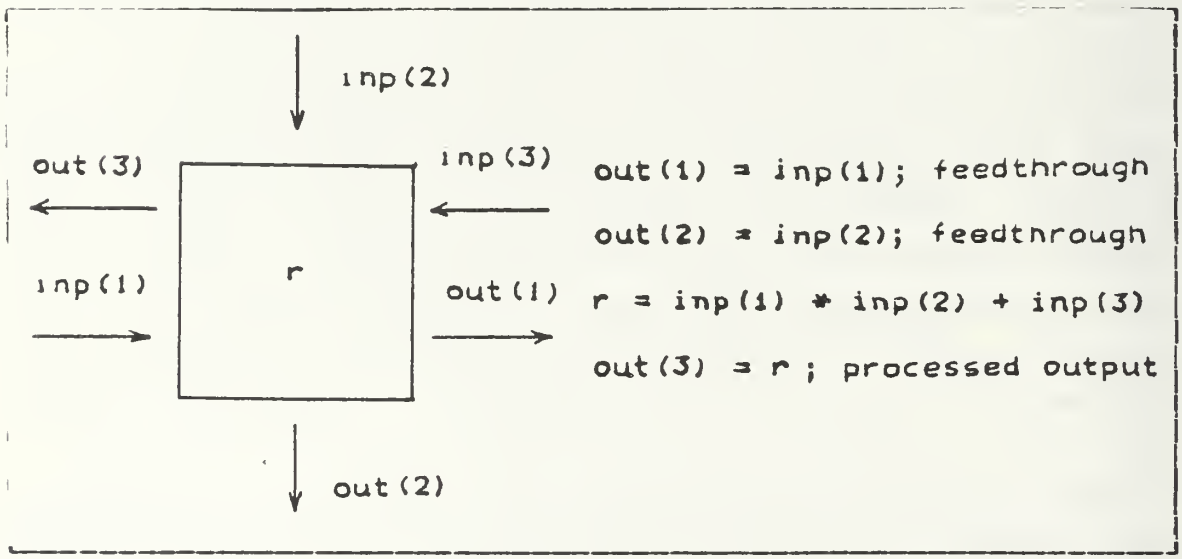



Figure 4.34 Inner Product Step Processor Cell

matrix and D is a a n-vector, both being given. To compute the vector X, we can use Forward Substitution as follows:

```
k := 1
y(i,1) := 0
```

```
repeat
     y(i,k+1) := y(i,k) + r(i,k) . x(k)
     k := k + 1
until k = i
x(i) := (d(i) - y(i,i)) / r(i,i)
```

The above algorithm can be used to calculate the elements of vector X in the sequence x(1), x(2), ... and so on. This algorithm can be implemented in a systolic array as will be shown. Y = (y(i,k)) is a vector of partial results that allows the recurrence to build up.

In our case, since the interest is in Back Substitution, we will simply enter the data into the systolic array in an order reverse to that proposed by Kung in his paper. Let us make this point more clear. We do Forward Substitution when we have a lower triangular array. In this case we solve first for x(1), next for x(2) and so on. Following the sequence proposed by Kung, vector X should be pumped into the systolic array as x(1), x(2) and x(3). The same for vector QB. Since here we are doing Back Substitution, we enter the vector X in a reverse sequence x(3), x(2), and x(1). QB is also entered the same way. The way matrix R being pumped into the systolic array is modified with respected to that proposed by Kung. For instance, the first of its elements to be pumped into the systolic array is r(3,3) (see cell 10 in Fig. 4.36) which is required to compute x(3). In the Forward Substitution Method, the first element pumped should be r(1,1) to compute x(1). The other elements of R are rearranged accordingly.

The system equation to be solved in this simulation is

$$RX = QB$$

where R is an upper triangular matrix and QB is a n-vector that resulted from the transformation seen in the previous Section.

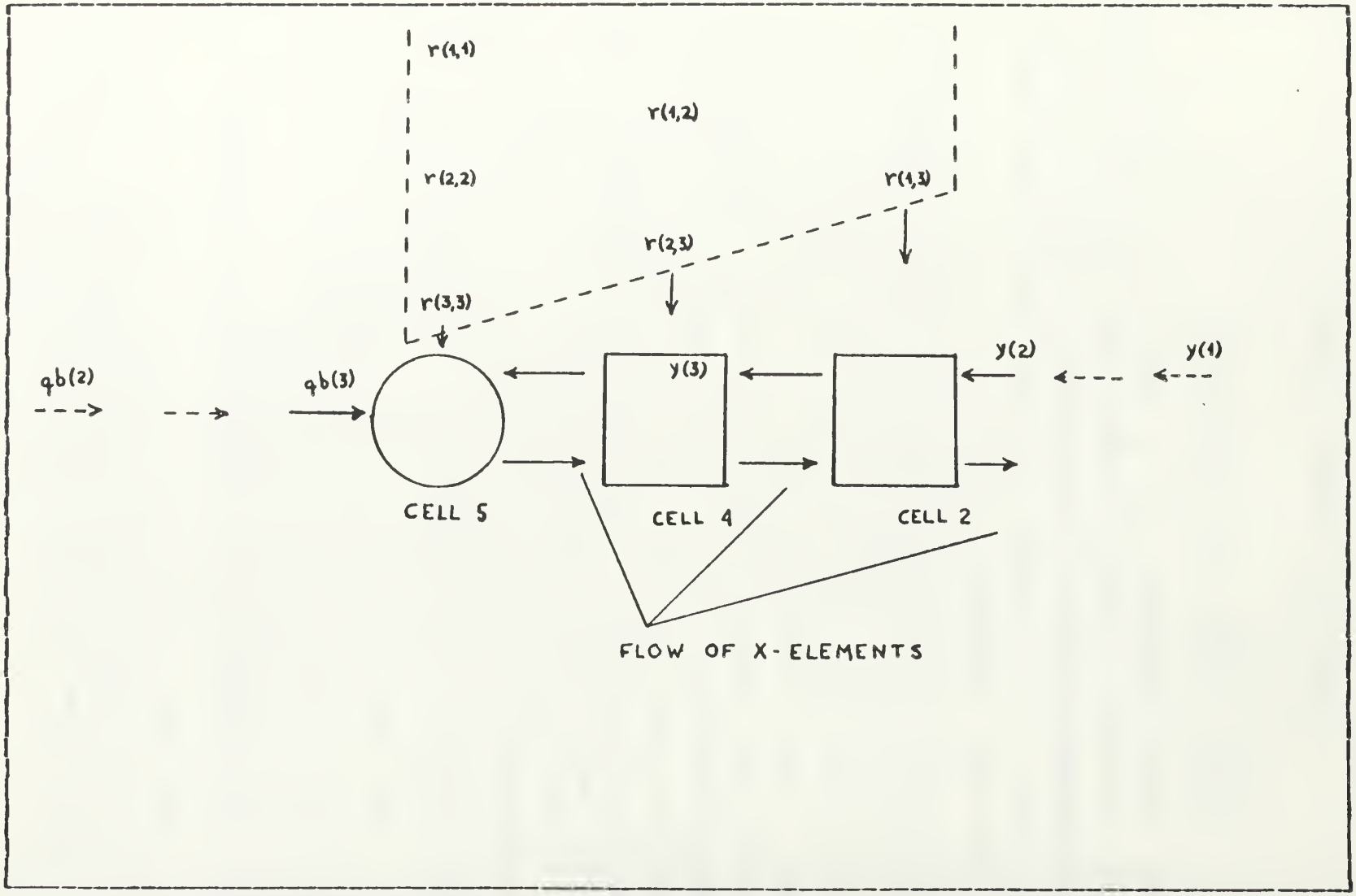



Figure 4.35 Data Arrangement for Back Substitution

Figure 4.35 presents the arrangement of the data with respect to the systolic array. The data elements are drawn in such a way that the required data synchronization to perform the operation is evident. Vector QB and matrix R actually carry data into the systolic array for processing. Vector Y enters the array bringing no data. Its elements are all zeros at the moment represented in Fig. 4.35 . Its values are modified as it flows through the array. Vector X is generated into cell 05, the Back Substitution Cell, and its elements are pumped through the array in a direction opposite to that of vector Y. As they go through cells 04 and 02, they combine with elements from matrix R to build up the elements of vector Y. Finally, X will come out with its final value when it leaves the array, from cell number 02 of Fig. 4.35.

Figures 4.35 and 4.36 should be compared. This last one presents the same arrangement as the former, but it shows how the picture appears on the screen. Matrix R is on the upper block of the cells (cells number 07 to 21) (Refer to Appendix C to compare the numerical values). This block does not belong to the array itself and is used only for presentation of the data. The systolic array is represented by cells 05, 04, and 02. The elements of vector QB are introduced from its left side (at cell 06). The vector of partial results Y, a string of zeros separated by one clock cycle delay, is pumped in from the right side (at cell 03). The output that will collect the solved elements of vector X is represented by cell 01. Attention to the fact that the numbers that are displayed at cells 04 and 02 are the values assumed by the elements of vector Y as they flow leftwards through the array. The number displayed in cell 05 is the element of vector X being computed and that in cell 01 is the element of X being pumped out to the external world. Another point to notice is the way the bidirectional

connections between cells 05 and 04, and 04 and 02 is implemented by SYSGRAS. Only one bidirectional arrow is used.

We decided to associate the elements of each column of the matrix R (third column in blue, second in red, and first in green) and the corresponding row elements of vectors X and QB with the same color. (e.g. the element number 3 of the QB will interact only with the column number 3 of the matrix R). Cell processor number 5, the so called Back Substitution Cell is shaped differently, as mentioned earlier, to emphasize its special purpose in the systolic array. This cell processor computes the elements of vector X from the received data and pumps them out to cell 04 to be utilized for calculation of the previously referred partial results. When the X elements complete their trip through the array, they are pumped out at cell 01.

The manner in which color is used to provide information is now described. The elements of vectors QB, X and Y, as well as matrix R, have primary colors (red, blue and green). When they meet elements of different primary colors, the cell where this meeting takes place assumes a secondary color that is the result of that combination. The elements of vector Y, for example, can be seen with their original color in cell 03, before mixing up with others. During their trip through the array they keep that color, but as a result of their combination with different color elements, the cell where they might be may present different secondary colors. However, we should keep in mind that only primary colors travel from a cell to another. Secondary colors are static.

To make these points more clear, we will track the formation of elements x(3) and x(2). Since R is an upper triangular matrix, we have

$$x(3) = qb(3) / r(3,3)$$

Substituting numerical values, as shown in Appendix C,

$$x(3) = -10.593 / 0.842 = -12.571$$

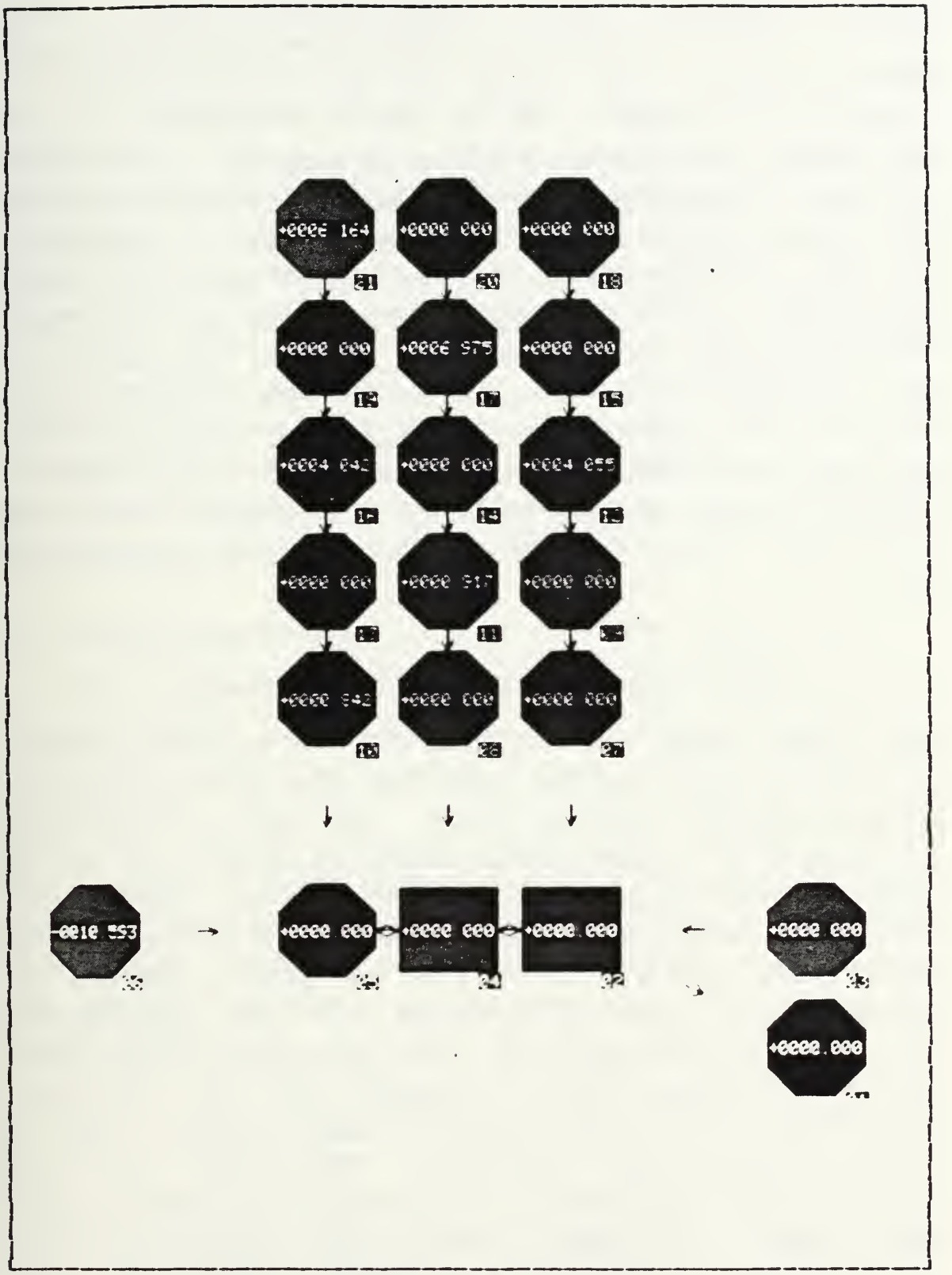


Figure 4.36 Back Substitution Array Initialization

As it has been pointed out, the Back Substitution Cell, cell number 05 in Fig. 4.37, is responsible for computing the elements of X. That Figure presents the computation of element x(3). Element gb(3) was pumped in from cell 06 (see Fig. 4.36) and element r(3,3) from cell 10 (see same Figure). It can also be seen in Fig. 4.37 that x(3) is blue since it was formed by the combination of gb(3) and r(3,3), both blue (remember that the third column of R has been coded blue). After its computation, element x(3) is pumped through the array to cell 04 (clock cycle 2, Fig. 4.38), to cell 02 (clock cycle 3, Fig. 4.39) and finally to the external world (clock cycle 4, Fig. 4.40) at cell 01. During this trip, its value is used for computation of the elements of vector Y which are necessary for computation of the other elements of vector X. Now let us track the formation of x(2).

From algebra, we have

$$x(2) = (gb(2) - x(3).r(2,3)) \;/\; r(2,2)$$

The partial result x(3).r(2,3) is computed at clock cycle 2, in cell 04. This cell receives x(3) from cell 05 (computed at clock cycle 1, see Fig. 4.37) and r(2,3) from cell 08 (see Fig. 4.37). This partial result is called y(2) (color coded red because it will contribute to the formation of x(2), which color code is red). This time, y(2), being originally red, appears in cell 04 as magenta because the contribution of y(2) is activated blue when it interacts with blue r(2,3) and blue x(3). At this point of the computation all necessary data to compute x(2) is available. Element y(2) is stored in cell 04, element gb(2) is ready to enter the array (see cell 06, in Fig. 4.38), and so is element r(2,2) (see cell 10 of Fig. 4.38). At clock cycle 3 these elements are pumped into the Back Substitution Cell (cell 05) (see Fig. 4.39) and the computation of x(2) takes

place. Substituting numerical values (as seen in those cells) into the above equation it becomes

$$x(2) = (-0.566 + 11.538) / 4.042 = 2.714$$

Element x(2) assumes color red because all factors that contributed to its formation had that color code. After its computation, x(2) is pumped through the array via cell 04 (clock cycle 4, Fig. 4.40), cell 02 (clock cycle 5, Fig. 4.41) and finally pumped out to cell 01. During this trip, similarly for x(3), it contributes to the formation of y(1), another element of the vector of partial results. This will be required to the computation of x(1).

The sequence of pictures from Fig. 4.36 to 4.44 displays the whole computational process in the systolic array according to the algorithm.

## E. FURTHER EXPLORATIONS

The potential of SYSGRAS goes far beyond the implementation of systolic algorithms. Presently, because of pratical importance, extensive research is being carried out on the investigation of faults in the actual systolic devices [Ref. 11]. How to circumvent those faults and which effect they do have on the results are some of the subjects under study. SYSGRAS can be used as a valuable tool in performing such studies, because of the simplicity of its interface with the user and because of the possibility of inclusion of possible subroutines to support the algorithms. To emphasize its versatility, we address the fact that a processor cell may have a large number of input/output ports (although it is presently set up for a maximum of four). This allows a large number of connections with other cells or the outside. This characteristic can be used, as example, to inject data into a cell that is situated in any position in the array.

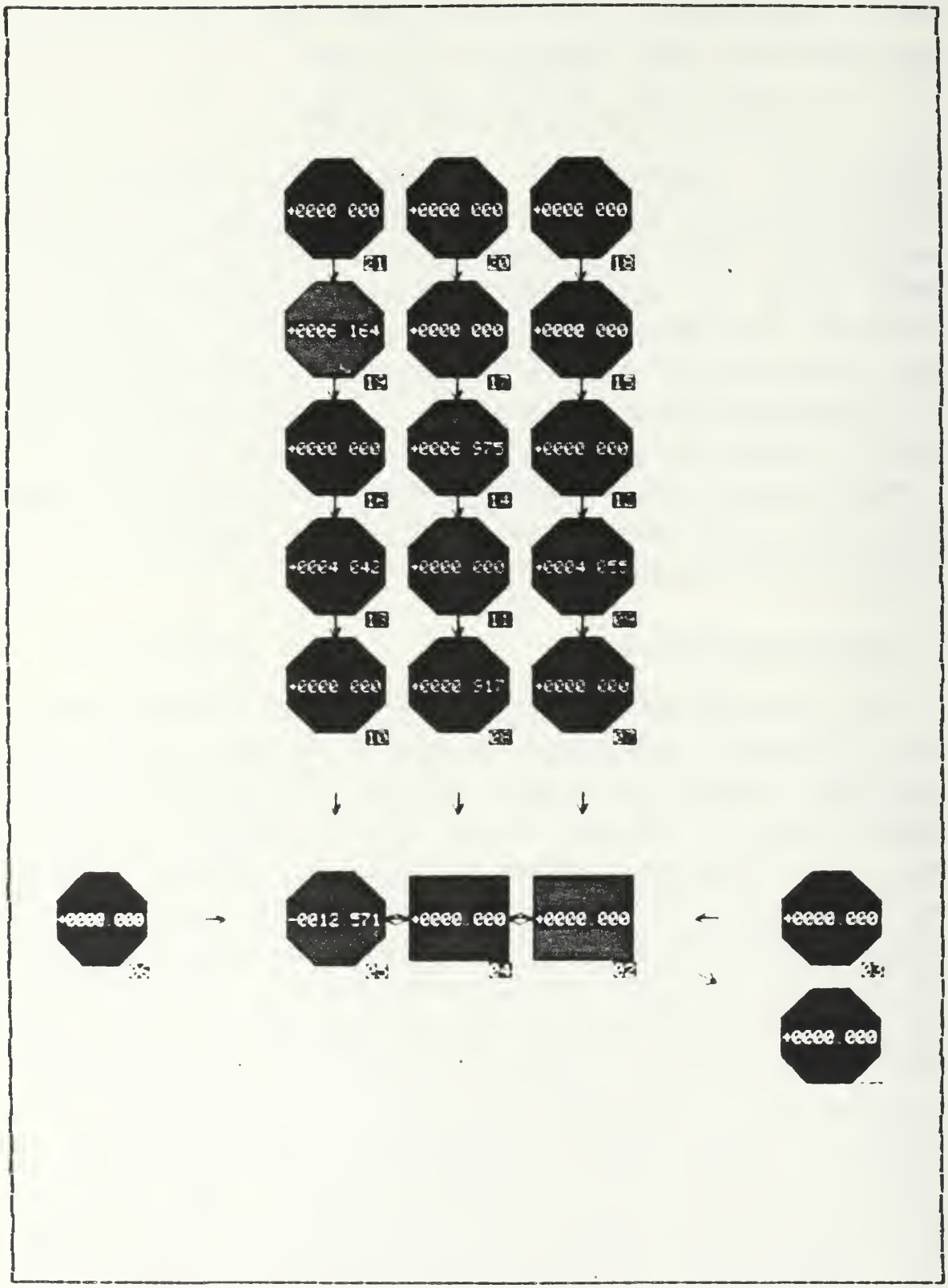


Figure 4.37 Back Substitution Array after Clock Cycle 1

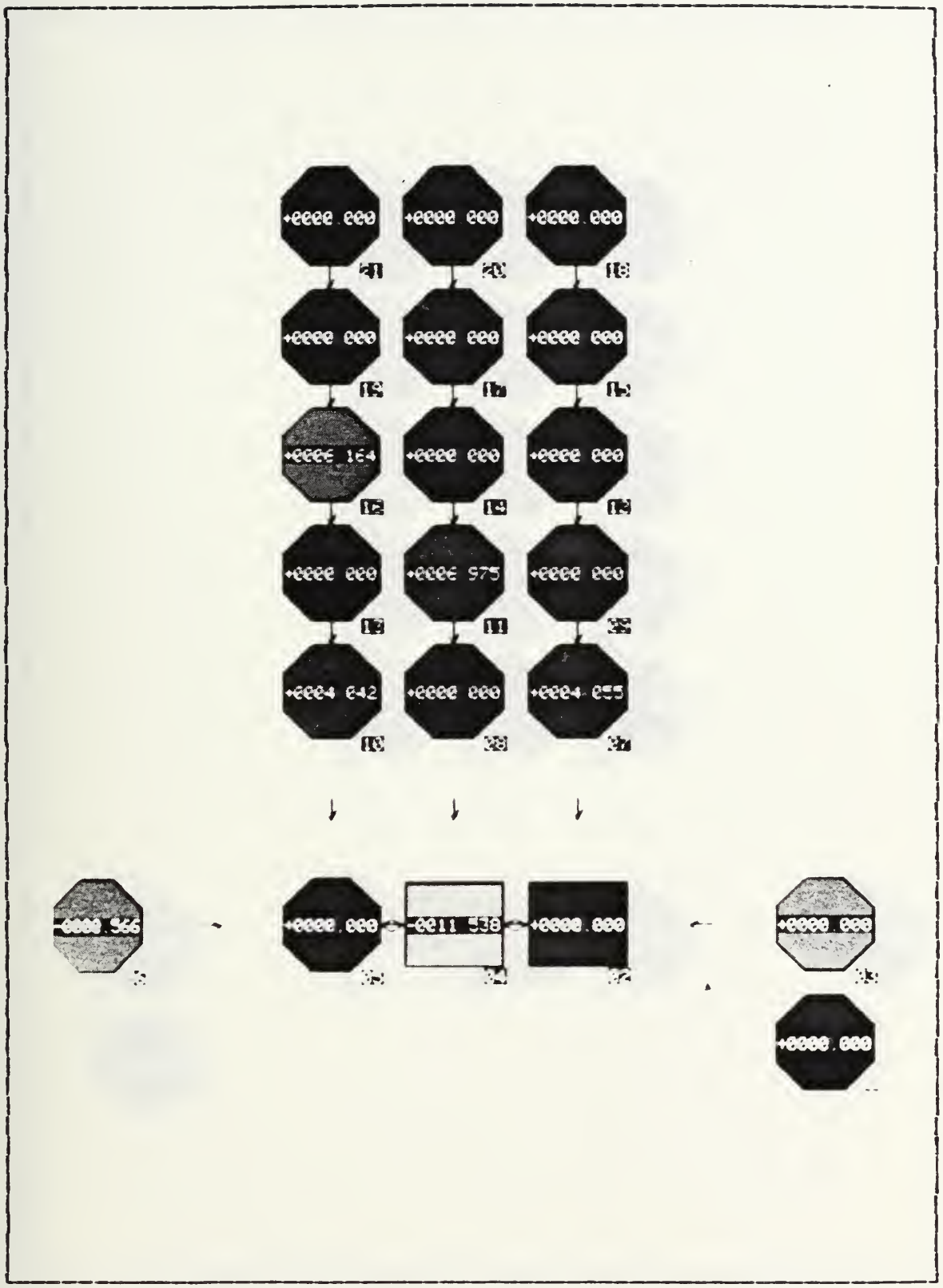


Figure 4.38 Back Substitution Array after Clock Cycle 2

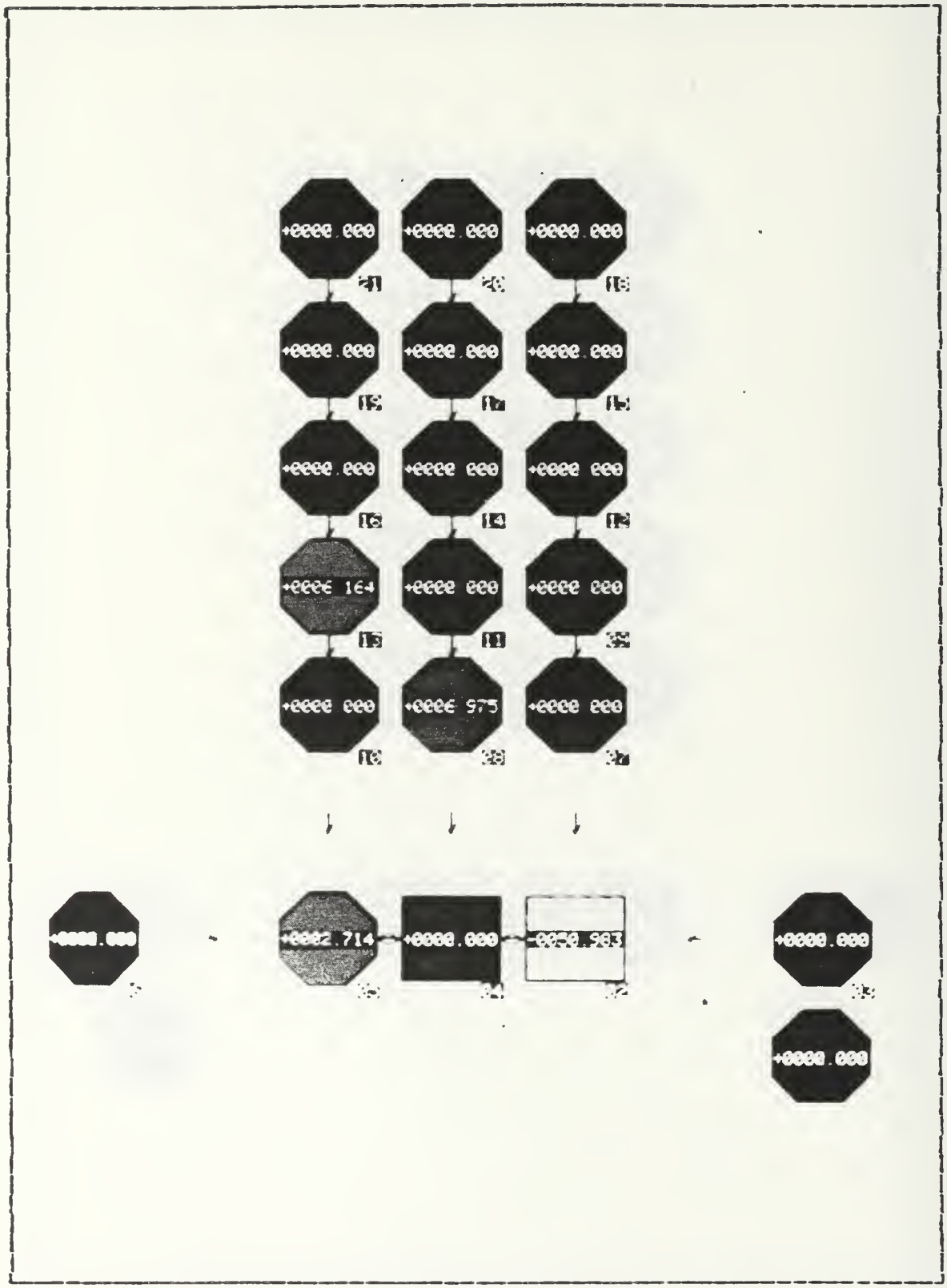



Figure 4.39 Back Substitution Array after Clock Cycle 3

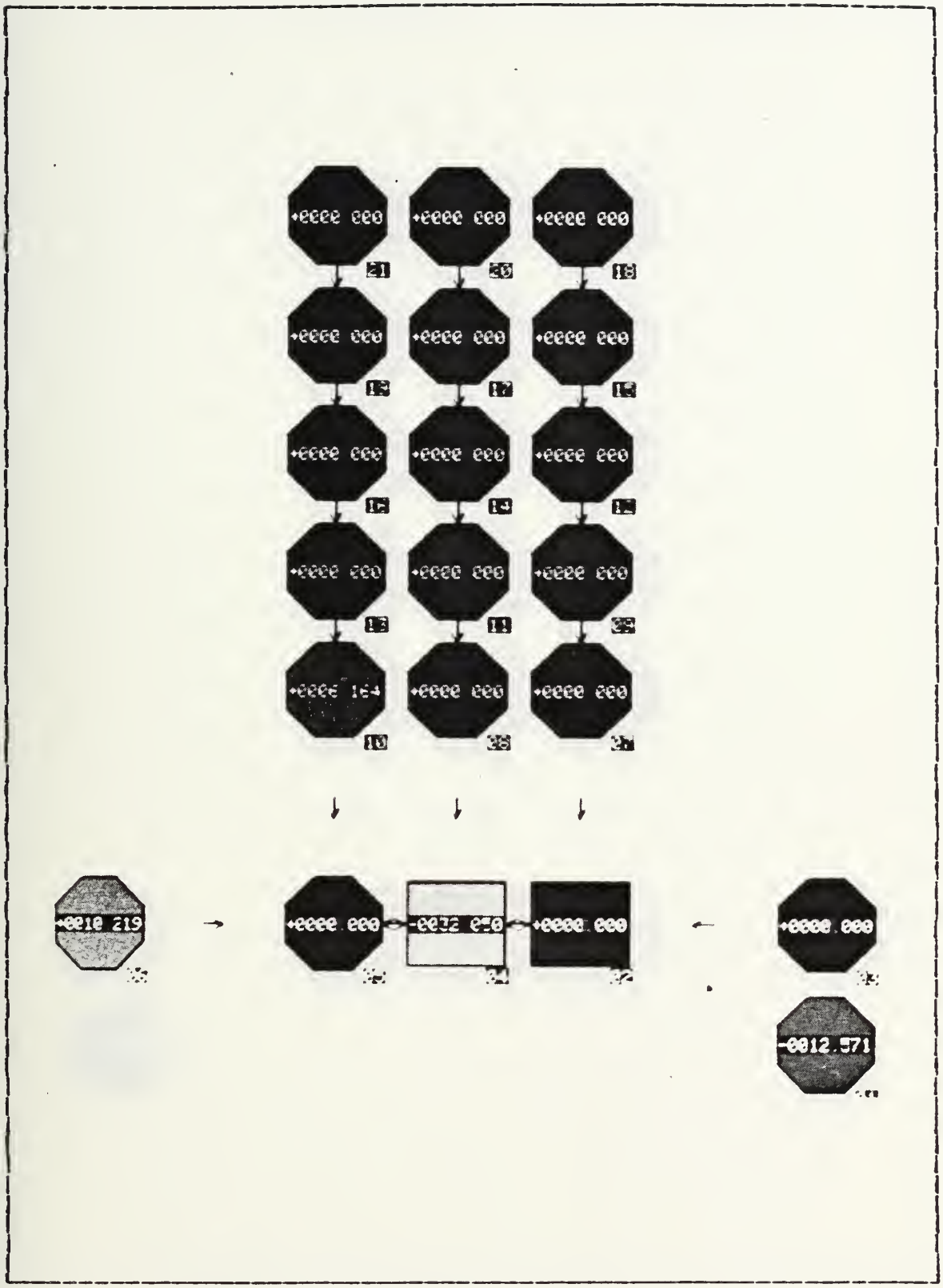


Figure 4.40 Back Substitution Array after Clock Cycle 4

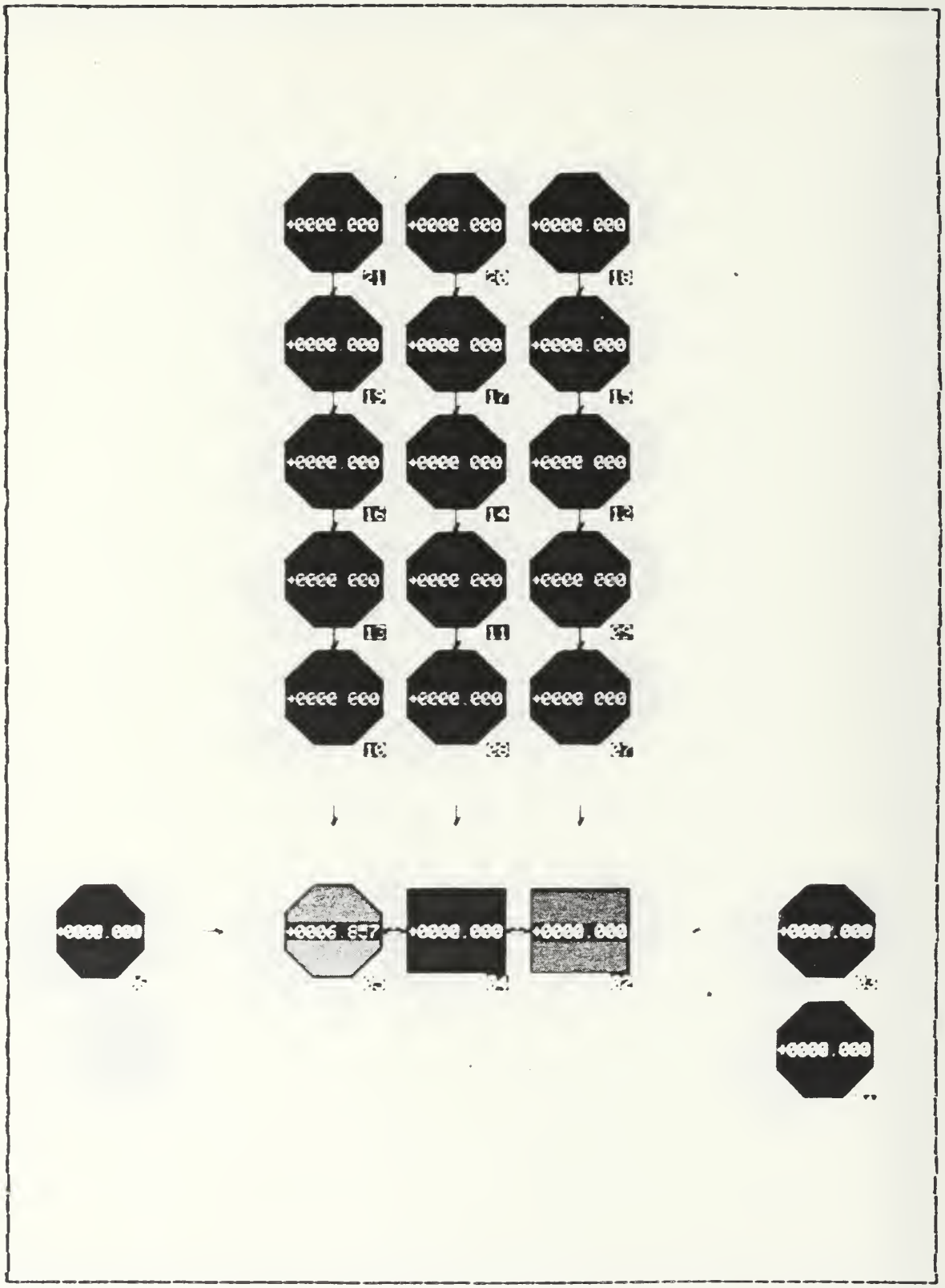


Figure 4.41 Back Substitution Array after Clock Cycle 5

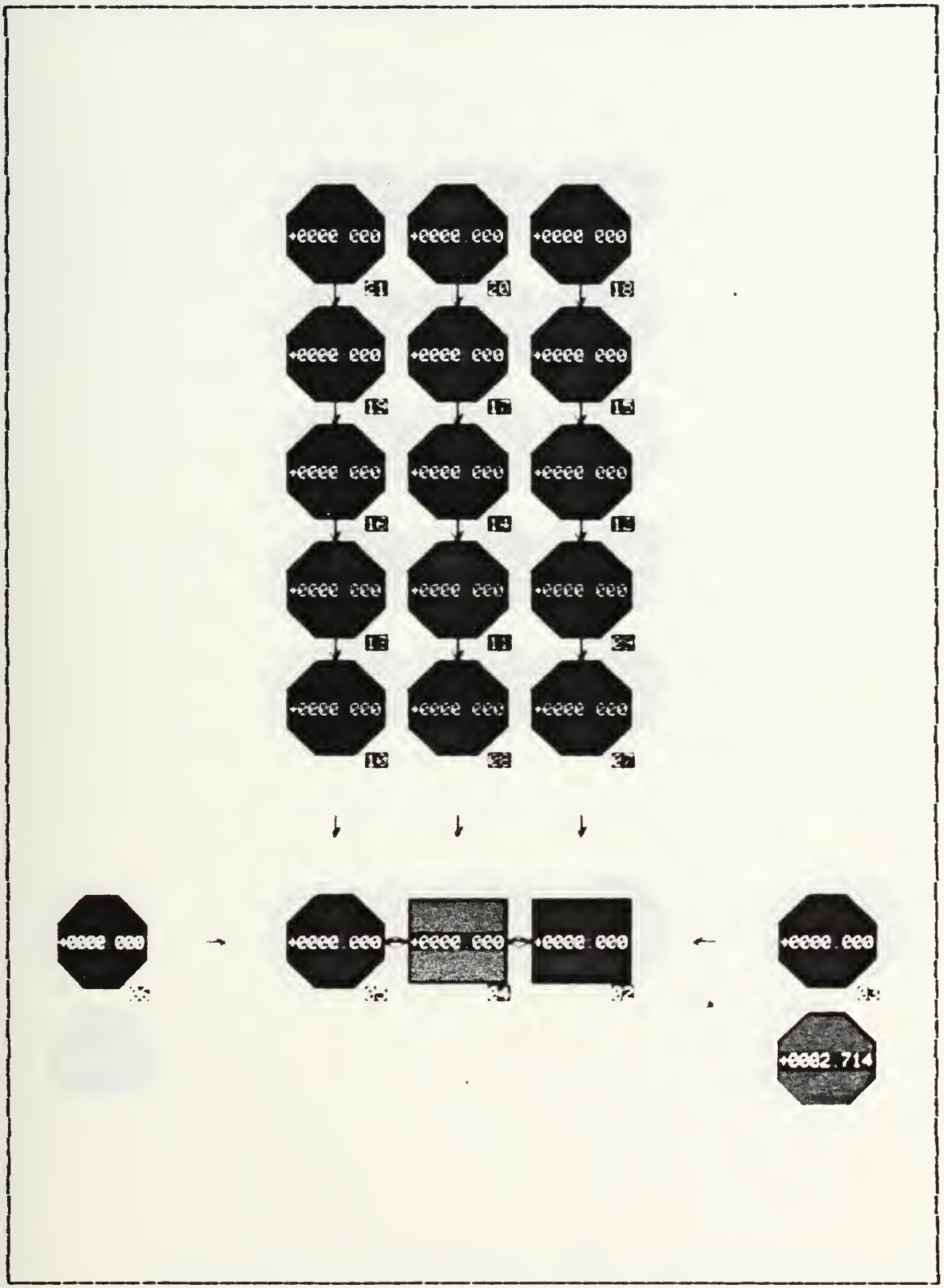



Figure 4.42 Back Substitution Array after Clock Cycle 6

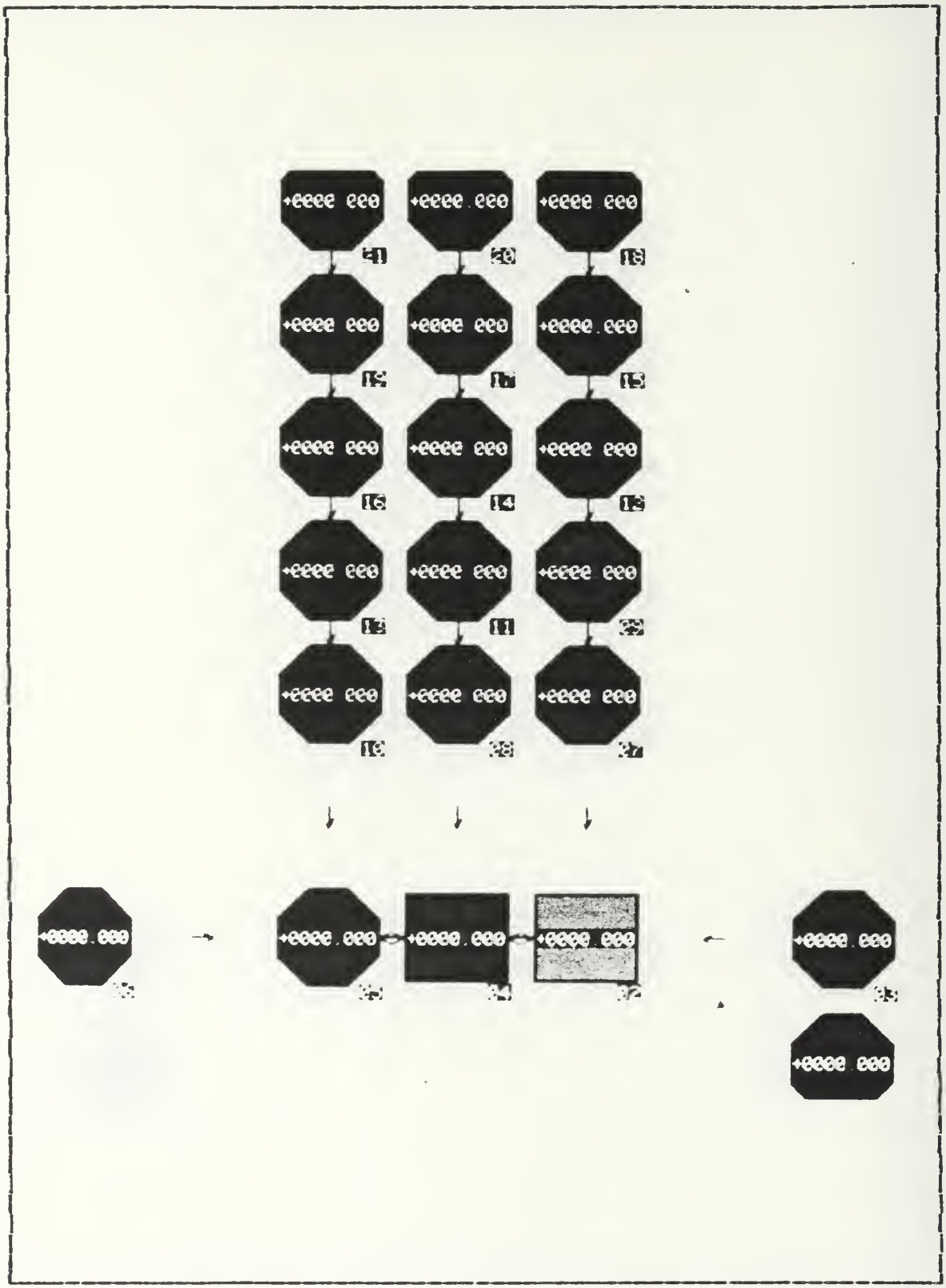


Figure 4.43 Back Substitution Array after Clock Cycle 7

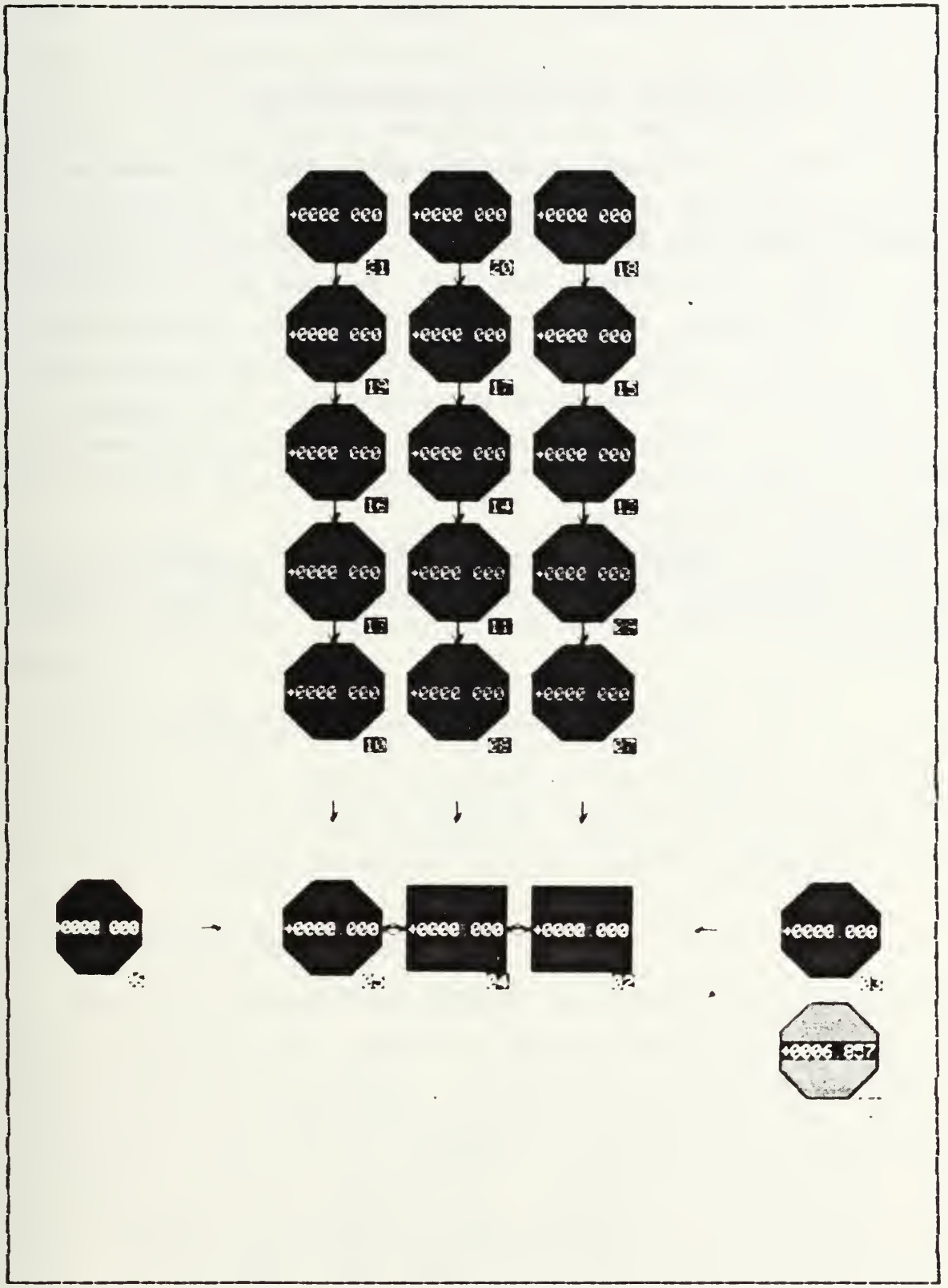


Figure 4.44 Back Substitution Array after Clock Cycle 8

These data could consist of logical commands that would alter the function of the processor for the selected cell. These commands, in a particular application, could "kill" the cell, or degrade its performance according to a desired need. Another approach, that would reduce the data entry volume, would be to design a subroutine to implement a defective processor and assign that processor to the target cell. The color feature can also be used to analyse the effect that faults at a particular cell will have over the others. If a color is injected at selected input ports (depending on the software of the subroutine that supports a processor), it will spread over the cells that receive data from this faulty cell under evaluation, and will display the "bad sector" on the screen.

Many other possible studies related to systolic arrays may be considered due to the flexibility of the SYSGRAS software.

## V. THE INTERACTION WITH THE SIMULATOR

### A. SOFTWARE REQUIREMENTS AND PROCEDURES

In order to operate with SYSGRAS, the user must have the following files.

SYSTOY.PAS
SYSFOR.FOR
LOGIN.COM
JUNK.COM

The user, after compiling SYSTOY.PAS and SYSFOR.FOR, must link both relocatable codes with SIGGRAPH core, which is at CS3202.CORE library. To do this, the command to be used is

LINK SYSTOY,SYSFOR,(CS3202.CORE)CORE/LIB

In order to run, enter the command "RUN SYSTOY". The code will address the RAMTEK peripheral system when running, and this device should be turned on and ready to operate.

### B. ENTERING A NEW PROBLEM

When the command "RUN SYSTOY" is entered, the simulator starts prompting all questions that must be answered by the user to enter the problem. In Appendix D we have recorded a session of the Simulator to make the interaction more understandable. It was our original intention to present this recording of the session as a guide to SYSGRAS for the Matrix Triangularization problem.However, due to the large amount of material that needs to be presented, we decided to include a short "dummy" session in Appendix D for clarity.

It shows how to enter the data and how to handle an incorrect input. The simulator permits recovery from entry errors and permits display of input data as many times as desired. The user has the following choices when dealing with the simulator:

- select a new problem run or a previous problem session
- record the session in a separate file for later printing
- ability to interrupt the graphics presentation at the end of each clock cycle

It is recommended that the user follow exactly the same nomenclature that was mentioned in Chapter III Section E to avoid errors that might be difficult to troubleshoot.

## C. REVIEW OF A PREVIOUS SESSION

We have included in Appendix E a recording of a session with SYSGRAS for the purpose of reviewing a previous session. The user must take care to rename the file which contains the data from previous session into a file under the name SYSARRAY.MEM . This file must be the most recent version generated by the VAX VMS Operating System.

The following files with data from previous sessions are already available, and may be run upon request from the user:

- file DIVDIFF.MEM, with recording of simulation analysed in Chapter IV, Section A.
- file MULTPLY.MEM, with recording of simulation analysed in Chapter IV, Section B.
- file GIVENSW.MEM, with recording of simulation analysed in Chapter IV, Section C.

- file BAKSUBS.MEM, with recording of simulation analysed in Chapter IV, Section D.

## D. RECORDING OF COMPUTED DATA AND PRINTING OUT

If the user has asked for this facility, SYSGRAS will write all computed results of each cell at each clock cycle into a file named SYSPRINT.DOC, which can be printed out after the end of the session.

# VI. THE SIMULATOR MAINTENANCE

## A. STRUCTURE OF THE SIMULATOR

The SYSGRAS has been designed in two layers. The upper layer that interacts with the user is in PASCAL, and so it is quite portable. In fact, due to the use of the "OTHERWISE" feature offered by the "CASE" command in the VAX PASCAL COMPILER, a few of its case structures have to be slightly modified before it can be installed in another machine. The lower layer is constructed in FORTRAN 77 and it is compiled with VAX FORTRAN. This layer interfaces with the Graphics Package SIGGRAPH that is presently available at NPS on the VAX 750 machine. It has been set up for presentation on the RAMTEK 9400 system, and so it is not portable. Its structure, however, can be modified to interface with another graphics package with characteristics similar to those of SIGGRAPH.

The PASCAL layer is presented in Appendix A and the FORTRAN layer is in Appendix B.

## B. MODULAR DESIGN ASPECTS

The design of SYSGRAS has followed the philosophy of modular design. The data structure has been designed to match the abstract idea of a systolic array and its multiple features, subsequently it can be easily identifiable by the program user. The basic element is the record "NODE". This element keeps all the information regarding each cell. The whole array is a collection of NODES's, and these are assembled into a higher level hierarchy by the record "GRAPH" which contains all information about the whole array at each clock cycle.

The part of the program that is most relevant tc the user refers to the cell processor support routines. If new types of cells need tc be simulated, new support subroutines must be added to the existing set. This can be done without complete knowledge of the program implementation because of the mcdularity of the design. Now we will refer specifically to this kind of software maintenance.

## C. DESIGNING AND INSTALLING A NEW PROCESSOR SUPPORT SUBRCUTINE

The cell processcr support subroutines modify the global data structure represented by the variable G of type GRAPH. The existing set of such subrcutines are under the comment titled "library routines for cell processors" in Appendix A. If a new one needs to be created and added, it should conform with the pattern seen in examples. The subroutine implementation must be placed between the statement "with G.NODES(.I.) do begin" and the statement "COLCR_PROCESSING(G,I)". This last command calls a subrcutine that will compute the color of the cell as a result of the ccmbination of the different primary colors of the input data.

The following steps must, therefore, be followed to introduce a new cell subroutine:

i. increase the constant NUMBER_OF_ROUTINES.

ii. modify the enumerated type PROCESSOR_TYPE to include another routine reference name (e.g. ROUTINE_9).

iii. modify procedure DEFINE_NODES to include references to the new subroutine. This should be done in such a way that the new routine is referred the same way as the others.

iv. modify procedure OUTPUT_ARRAY the same way as above.

v. modify procedure CORRECT_NODE as above.

vi. modify the main program statement "case PROCESSOR cf" as above.

vii. place the body of the new procedure next to the existing procedures to conform with the program structure.

Hints for the design of the new procedure:

- the type NODE defines the cell data structure.
- the number of input/output ports in a cell can be modified by changing the constant CONNECTIONS in the main program. If this is done, the constant MAXLINKS also must be modified as instructed in the program text comments.

The present implementation of SYSGRAS restricts the maximum number of cells in an array to 23, in order to achieve a faster computational speed. If desired, that can be modified by altering the main program constant MAXNODES. If this is done, the constant MAXLINKS must also be modified as instructed in the program text comments.

# VII. CONCLUSIONS

## A. ESTABLISHING COMPARISON PARAMETERS

The simulations that have been studied in Chapter V provided us a reasonable tool to use for evaluation of algorithms implemented in systolic arrays. Important factors that must be considered in this kind of an evaluation are:

i. use of the same type of processors in other algorithms

ii. average percentage of cells involved in effective computation per clock cycle

iii. degree of homogeneity of cells

iv. degree of the usage of pipelining

v. possibility of modular expandability

vi. required number of external connections to each cell

vii. cell complexity

Unfortunately not all these factors can be determined by using this tool. Subjectiveness also has to play a role in this evaluation.

## B. CONCLUSIONS ABOUT ALGORITHMS UNDER ANALYSIS

The implementation of an algorithm in a systolic array may provide a greater calculation speed, but a cost was paid to get that achievement. The cost can be evaluated in terms of the complexity of the design, the difficulty in implementation, the manufacture problems that can result from a particular array configuration, etc. We want to make sure

that this cost is not excessive and, ideally, to be minimum. These are considerations that require criteria of evaluation to be established. Instead of trying to come up with a criteria that could be the subject of lengthy discussion, the above mentioned factors are used to discuss the efficiency of the algorithms with a more qualitative than quantitative approach.

The first factor we want to establish is that concerning the average percentage of cells effectively involved in active computation per clock cycle in each algorithm. For each algorithm we consider the number of required cycles to run till completion, the number of physical cells in the systolic array, and the number of cells involved in active computation in each clock cycle. This last factor corresponds to the number of cells shown on the screen with a color other than black. Thus we do have:

Matrix Triangularization:
number of cycles = 9
total number of cells = 9
percentage of computation / cycle =
= 1/9 x 1/9 x (1+2+4+5+6+5+3+1+0) =
= 0.333

Performing similar calculations for the other algorithms, we obtain the numbers shown up in Table I. Definitely, the Matrix Triangularization has a longer "duty cycle" per cell and this means less waste.

The number of types of cells required for each algorithm is easily seen in the pictures shown in Chapter IV. Modular expandability is another important feature. It implies the possibility of interconnecting chips (each one embedding a whole systolic array) in a cascade fashion or some other arrangement where the chips are used as smaller parts of a higher hierarchical arrangement. Matrix Triangularization

and Back Substitution have restrictions in that respect due to the fact that their cell arrangement is not symmetrical and such an expansicn would require different types of chips.

With respect to these factors, we can summarize in the following table:

TABLE III

Ccmparison of Algorithm Implementation Evaluation Factors

| FEATURE | MATRIX TRIANG. | BACK SUBST. | DIV. DIFF. | MATRIX MUIT. |
|---|---|---|---|---|
| Average percentage of duty cycle ccmputing per cell | 0.333 | 0.250 | 0.250 | 0.153 |
| Number of cell types | 2 | 2 | 1 | 1 |
| Modular expandability | R | R | Y | Y |

R = restricted
Y = yes

The number of external connections required by each cell is pehaps the most demanding factor existing in pratical implementations. The normal ccnnections are power and clock lines. Some algorithms like Matrix Multiplication need only these twc. Some others have synchronization and data feeding problems that can only be solved with the addition of extra external connections to the cell. This feature is somehow related to the degree of pipelining that can be achieved.

The Matrix Triangularization and the Divided Differences algorithms can be pratically implemented if there is a flag signal that can trigger a pumping out mechanism to send the data frozen in each cell after the last clock cycle. To recover the data and display it externally, additional connecticns are needed in each cell. This increases the total ccst of the chip and complicates the problem of modular expandability. Since a design goal in every VLSI implementation is to reduce the number of external connections, this is a key factor in the design effort. Pipelining possibility is also affected because the pumping out operation is not part of the normal algorithm cycle. It is an additional burden that may require the interrupticn of the data pumping intc and the pipelining will have to be restricted to data of the same problem. Data from different problems can not coexist at the array at the same time. A calculaticn of a prchlem has to finish in order to start another. In this sense, if we examine the Matrix Multiplication algorithm, we will conclude that true pipelining can be achieved. No data flow interruption occurs because the results are not kept frozen in the cells, but are moved as part of the data flow.

The factor of cell complexity is important not only because cf the possible high cost of the hardware, but also because cf the time that may be required for a clock cycle to be executed. If a longer time is required, the clock speed has to be kept low and the efficiency of the processing decreases. Simple operations are always a goal on the algorithm design because they result simpler and faster hardware. This is why another algorithm for Matrix Triangularization without Square Roots has been suggested.

## C. SUGGESTIONS FOR FUTURE MODIFICATIONS ON SYSGRAS

We are conscious that we have not achieved an optimal design with our simulator in terms of performance. It could be improved if the graphics implementation were more efficient. This would speed up the interaction with the user and make it more attractive. Another point that could be improved is the user interface. The amount of information that is required from the user requires an effort that might be minimized if other interface techniques were used, such as combined use of mouse or lightpen with the keyboard input.

Additional points that could be modified to enhance the presentation at the screen are:

- Elimination of non significant zeros from the cell display (such as turning 000.400 into 0.4).
- Change of the bidirectional arrow that represents communications in both directions between cells into unidirected arrows, one for each link between cell ports. This would make the cells appear on the screen the same way they are represented in the literature. An example of the birectional arrow can be seen in Fig. 4.36, in Chapter IV, connecting cells 05 and 04, as well as 04 and 02.

## D. THE ROLE OF THE SIMULATOR

Our goal was to contribute to the understanding of the implementation of algorithms in systolic arrays. This led us to the realization of some of the inherent problems that appear in this field. The complexity of the data interaction in the space-time frame is considered to be the greatest obstacle in the understanding process. To decide on the approach to adopt to handle the problem is also difficult. In response to that, we designed a tool whose

task is to provide a software implementation of the systolic array. The user is left free to concentrate on the algorithm. We have shown the use of this tool on the study of some algorithms. This gave us the opportunity to realize the power of a systolic array in the process of perfcrming calculations. Aspects such as computation time, memory requirements and I/O interactions are minimized. The mapping of an algorithm onto a systolic array certainly represents a major difficulty. The main role of our simulator is not to help in this mapping, but it can be used in the validation phase of the algorithm design. The greatest contribution that we see in SYSGRAS is its versatility. The user can model an ideal envircnment for the algorithm or an environment contaminated by problems in the underlying hardware. This certainly can help in achieving better results while presenting a clear idea of the robustness of an algorithm.

# APPENDIX A

## SYSTOLIC ARRAY SOFTWARE SIMULATION PROGRAM

```
program SYSTOLIC_SIMULATION (input,output,cells,rec);

                                        (* version JAN/31 15:00 *)

        (* runs on VAX 750 under PASCAL VAX compiler *)

  label 100;
  const CONNECTIONS=4;
        MAXNODES=23;
        NUMBER_OF_RCUTINES=8;
        CELL_MEMORIES=4;
        MAXLINKS=92; (* = CONNECTIONS*MAXNODES *)
        PRIMARY_COLORS=4;
        MAXCOORD=8;
        NUMBER_OF_SHAPES=2;
  type  LEAD=1..CONNECTIONS;
        CCLORS=(BLACK,BLUE,GREEN,RED,MAGENTA,CYAN,YELLOW,WHITE);
        SHAPETYPE=(AMORF,SQUARE,OCTAGON);
        SHAPECODE=1..NUMBER_OF_SHAPES;
        CCLORCODE=1..PRIMARY_COLORS;
        TRANSFER=reccrd
                        DATUM:real;
                        SPECTRUM:COLORS
```

```
          end;
BUFFER_IN=array(.LEAD.) of TRANSFER;
MEMORY=array(.1..CELL_MEMORIES.) of real;
BUFFER_OUT=array(.LEAD.) of TRANSFER;
NODEPTR=1..MAXNODES;
RAINBOW=set of COLORS;
PROCESSOR_TYPE=(ROUTINE_0,ROUTINE_1,ROUTINE_2,ROUTINE_3,
                ROUTINE_4,ROUTINE_5,ROUTINE_6,ROUTINE_7,
                ROUTINE_8);
TASK_TYPE=(IMPORTER,INTERNAL,EXPORTER);
NODE=record
           INP:BUFFER_IN;
           MEM:MEMORY;
           OUT:BUFFER_OUT;
           COLOR:RAINBOW;
           PROCESSOR:PROCESSOR_TYPE;
           TASK:TASK_TYPE;
           XCOORD:1..MAXCOORD;
           YCOORD:1..MAXCOORD;
           SHAPE:SHAPETYPE
     end;
GRAPH=record
           NODES:array(.NODEPTR.) of NODE;
           ARCS:array(.NODEPTR,LEAD,NODEPTR,LEAD.) of
```

```
                                                        boolean
            end;
      ROUTNUMB=1..NUMBER_OF_ROUTINES;
      LINKTABLE=array(.1..4,1..MAXLINKS.) of integer;
var   G:GRAPH;
      I,K,NODESNUME,CELLNUMB,CELLORIG,CELLDEST:NODEPTR;
      INPUTKEY,L,INPORT,OUTPORT:LEAD;
      X,Y,TIMER,J,M,LINKNUMB:INTEGER;
      DIF_ROUT_NUME,ROUTINE_ID:ROUTNUMB;
      LINKLIST:LINKTABLE;
      COLORNUMBER:COLORCODE;
      CELLQUANT:0..MAXNODES;
      SHAPENUMBER:SHAPECODE;
      NODE_IS_WRONG,LINK_IS_WRONG,WANT_REC:boolean;
      CORRECTION_CODE,ECHOCODE,RECCODE,INPUT_CHECK:char;
      INTERRUPT_CICCK,GO_AHEAD:char;
      CELLS:file of GRAPH;
      ERRORCODE:integer;
      REC:text;

(* routines for displaying at screen of graphics device *)

procedure INITIO; fortran;
procedure CLRSCR; fortran;
procedure FINIT;  fortran;
```

```
prccedure SQUARED (%ref COLOR:integer; %ref COORDX:integer;
                   %ref COORDY:integer); fortran;
procedure OCTGON (%ref COLOP:integer; %ref COORDX:integer;
                  %ref COORDY:integer); fortran;
procedure FILCEL (%ref COORDX:integer;
                  %ref COORDY:integer; %ref NUMB:real); fortran;
procedure IDENT (%ref COORDX:integer; %ref COORDY:integer;
                 %ref ID:integer); fortran;
procedure ARROW (%ref X1:integer; %ref Y1:integer;
                 %ref X2:integer; %ref Y2:integer); fortran;

(* primitive operations on systolic array graph *)

procedure JOIN (var G:GRAPH; NODE1,NODE2:NODEPTR; LEAD1,
                                          LEAD2:LEAD);
 begin (* add an arc from NODE1/LEAD1 to NODE2/LEAD2 *)
  G.ARCS(.NODE1,LEAD1,NODE2,LEAD2.):=TRUE
 end;
procedure REMOVE (var G:GRAPH; NODE1,NODE2:NODEPTR;
                  IEAD1,LEAD2:LEAD);
 begin (* delete an arc from NODE1/LEAD1 to NODE2/LEAD2 *)
  G.ARCS(.NODE1,LEAD1,NODE2,LEAD2.):=FALSE
 end;
function ADJACENT (G:GRAPH; NODE1,NODE2:NODEPTR;
        LEAD1,LEAD2:LEAD): boolean;
```

```
 begin (* test whether there is an arc from NODE1/LEAD1 to
         NODE2/LEAD2 *)
  if G.ARCS(.NODE1,LEAD1,NODE2,LEAD2.) then
     ADJACENT:=TRUE
  else ADJACENT:=FALSE
 end;

(* utility routines *)

procedure CELL_IDENTIFICATION_NUMBER (ROUTINE:PROCESSOR_TYPE);
 begin
  writeln('ENTER NUMBER OF CELLS THAT WILL PERFORM');
  writeln('THIS RCUTINE');
  readln(CELLQUANT);
  writeln('ENTER IDENTIFICATION NUMBER FOR EACH CELL;');
  writeln('HIGHEST ID NUMBER IS ',MAXNODES:2,':');
  for I:=1 to CELLQUANT do
    begin
      writeln('ENTER CELL NUMBER ',I:2,' ID NUMBER');
      readln(CELLNUMB);
      G.NODES(.CELLNUMB.).PROCESSOR:=ROUTINE
    end
 end;
procedure DEFINE_NODES (var G:GRAPH);
(* define working nodes *)
```

```
var M,I: integer;
begin
 writeln;
 writeln;
 writeln('YOU WIIL BE PROMPTED TO CONSTRUCT A SYSTOLIC ARRAY');
 writeln('KEEP IN MIND THAT ONLY THE NUMBERS YOU WILL ASSIGN');
 writeln('TO THE CELLS WIIL BE KEPT AS IDENTIFICATION BY THE');
 writeln('SYSTEM;  LINKS ARE NOT  IDENTIFIED BY  THEMSELVES,');
 writeln('ONLY BY CELLS THEY CONNECT');
 writeln;
 writeln('ENTER TOTAL NUMBER OF CELLS; MAXIMUM IS ',MAXNODES:2);
 readln(NODESNUMB);
 writeln('NOW YOU MUST SPECIFY THE CELLS'' PROCESSING FUNCTION');
 writeln('AND TASK.');
 (* routine selection *)
 writeln('WE HAVE PRESENTLY ',NUMBER_OF_ROUTINES:1,
         ' ROUTINES IN OUR BUILT-IN LIBRARY');
 writeln('AND THEY ARE ',
         'IDENTIFIED 1 TILL ',NUMBER_OF_ROUTINES:1);
 writeln;
 writeln('ENTER NUMBER OF DIFFERENT ROUTINES YOU WANT TO USE');
 readln(DIF_ROUT_NUMB);
 for M:=1 to DIF_ROUT_NUMB do
   begin
```

```
      writeln('ENTER IDENTIFICATION NUMBER OF ROUTINE:');
      writeln('      GIVENS W/ SQRT (internal cell)  => 1');
      writeln('      GIVENS W/ SQRT (external cell)  => 2');
      writeln('      BUFFER CELL                     => 3');
      writeln('      INNER PRODUCT STEP              => 4');
      writeln('      GIVENS W/O SQRT (internal cell) => 5');
      writeln('      GIVENS W/O SQRT (external cell) => 6');
      writeln('      BACK_SUBSTITUTION (round cell)  => 7');
      writeln('      DIVIDED DIFFERENCE              => 8');
      readln(ROUTINE_ID);
      case ROUTINE_ID of
        1:CELL_IDENTIFICATION_NUMBER(ROUTINE_1);
        2:CELL_IDENTIFICATION_NUMBER(ROUTINE_2);
        3:CELL_IDENTIFICATION_NUMBER(ROUTINE_3);
        4:CELL_IDENTIFICATION_NUMBER(ROUTINE_4);
        5:CELL_IDENTIFICATION_NUMBER(ROUTINE_5);
        6:CELL_IDENTIFICATION_NUMBER(ROUTINE_6);
        7:CELL_IDENTIFICATION_NUMBER(ROUTINE_7);
        8:CELL_IDENTIFICATION_NUMBER(ROUTINE_8);
      end;
      if (M < DIF_ROUT_NUME) then
        writeln('OK FOR THIS ROUTINE; LET''S SEE ANOTHER');
    end;
  writeln('ENTER NUMBER OF CELLS THAT WILL RECEIVE EXTERNAL DATA');
```

```
readln(CELLQUANT);
writeln('ENTER IDENTIFICATION NUMBER FOR EACH CELL; ',
        'HIGHEST ID NUMBER IS ',MAXNODES:2);
for I:=1 to CELLQUANT do
 begin
   writeln('ENTER CELL NUMBER ',I:2,' ID NUMBER');
   readln(CELLNUMB);
   G.NODES(.CELLNUMB.).TASK:=IMPORTER
 end;
writeln('ENTER NUMBER OF CELLS THAT WILL OUTPUT DATA TO ',
        'EXTERNAL WORLD');
readln(CELLQUANT);
if (CELLQUANT <> 0) then
 begin
  writeln('ENTER IDENTIFICATION NUMBER FOR EACH CELL; ',
        'HIGHEST ID NUMBER IS ',MAXNODES:2);
  for I:=1 to CELLQUANT do
    begin
      writeln('ENTER CELL NUMBER ',I:2,' ID NUMBER');
      readln(CELLNUMB);
      G.NODES(.CELLNUMB.).TASK:=EXPORTER
    end (* for *)
 end (* if-then *)
else writeln('OK, NO OUTPUT TO EXTERNAL WORLD');
```

```
writeln;
writeln('NOW YOU WILL ASSIGN SHAPE AND  POSITION  ON  SCREEN  TO');
writeln('EACH CELL THAT WILL CONSTITUTE  YOUR  ARRAY.  AVAILABLE');
writeln('SHAPES  ARE   OCTAGON   AND   SQUARE.   FOR COORDINATES');
writeln('ASSIGNMENT, THE SCREEN IS DIVIDED IN 64 POSITIONS  LIKE');
writeln('A CHESSBOARD AND LEFTMOST BOTTOM POSITION IS (X=1,Y=1).');
writeln('ABCISSA IS  X  AND IT GOES FROM  1 TO 8. ORDINATE IS Y,');
writeln('AND IT GOES FROM 1 TO 8 ALSO');
writeln;
writeln('ENTER ID NUMBER,SHAPE AND POSITION TO YOUR ',NODESNUMB:2,
        ' CELLS');
writeln;
for I:=1 to NODESNUMB do
 begin
   writeln('ENTER CELL NUMBER ',I:2,' ID NUMBER');
   readln(CELLNUMB);
   writeln('ENTER SHAPE:');
   writeln('              SQUARE  ==> 1');
   writeln('              CCTAGON ==> 2');
   writeln('  TAKE A CHOICE|');
   readln(SHAPENUMBER);
   case SHAPENUMBER of
      1: G.NODES(.CELLNUMB.).SHAPE:=SQUARE;
      2: G.NODES(.CELLNUMB.).SHAPE:=OCTAGON
```

```
        end (* case *);
        writeln;
        writeln('ENTER SCREEN COORDINATES:');
        writeln('      ENTER X (integer from 1 to 8)');
        readln(G.NODES(.CELLNUMB.).COORDX);
        writeln('      ENTER Y (integer from 1 to 8)');
        readln(G.NODES(.CELLNUMB.).COORDY);
        if ( I < NODESNUMB ) then
            writeln('OK, NOW FOR NEXT CELL');
      end (* for *);
     writeln;
     writeln;
   end (* DEFINE_NODES *);
procedure DEFINE_ARCS ( var G:GRAPH );
 (* define arcs *)
 var M:integer;
 begin
   writeln('NOW  YCU  WILL  BE  ASKED  TO  DEFINE  THE  LINKING ');
   writeln('BETWEEN CELLS; ATTENTION TO THE FACT THAT EACH CELL ');
   writeln('HAS ',CONNECTIONS:1,' PORTS FOR INPUT AND THE SAME ',
               'NUMBER FOR OUTPUT');
   writeln('YOU MUST SPECIFY  FROM  WHICH  CELL/PORT  TO  WHICH');
   writeln('CELL/PCRT THE LINKING MUST BE ESTABLISHED');
   writeln;
```

```
  writeln('ENTER THE NUMBER OF LINKS YOU DO WANT TO ESTABLISH');
  readln(LINKNUMB);
  for M:=1 to LINKNUMB do
    begin
     writeln('ENTER ID NUMBER OF ORIGIN CELL FROM LINK ',
             'NUMBER ',M:3);
     readln(CELLORIG);
     writeln('ENTER OUTPUT PORT NUMBER FROM ORIGIN CELL; ',
             'MAXIMUM IS ',CONNECTIONS:1);
     readln(OUTPORT);
     writeln('ENTER ID NUMBER OF DESTINATION CELL FROM ',
             'LINK NUMBER ',M:3);
     readln(CELLDEST);
     writeln('ENTER INPUT PORT NUMBER FROM DESTINATION ',
             'CELL; MAXIMUM IS ',CONNECTIONS:1);
     readln(INPORT);
     JOIN(G,CELLORIG,CELLDEST,OUTPORT,INPORT)
    end;
end (* DEFINE_ARCS *);
procedure INITIALIZE_GRAPH (var G:GRAPH);
 (* initialize graph *)
 var I,J,K,L:integer;
 begin
   (* initialize nodes *)
```

```
  for I:=1 to MAXNODES do with G.NODES(.I.) do
    begin
      for J:=1 to CONNECTIONS do
        begin
          INP(.J.).DATUM:=0.0;
          INP(.J.).SPECTRUM:=BLACK;
          OUT(.J.).DATUM:=0.0;
          OUT(.J.).SPECTRUM:=BLACK;
        end;
      for J:=1 to CELL_MEMORIES do MEM(.J.):=0.0;
      COLOR:=(.BLACK.);
      PROCESSOR:=ROUTINE_0;
      TASK:=INTERNAL;
      SHAPE:=AMORF;
      COORDX:=1;
      COORDY:=1;
    end;
(* initialize arcs *)
for I:= 1 to MAXNODES do
  for J:=1 to CONNECTIONS do
     for K:=1 to MAXNODES do
       for L:=1 to CONNECTIONS do
           G.ARCS(.I,J,K,I.):=FALSE;
end (* INITIALIZE_GRAPH *);
```

```
prccedure OUTPUT_ARRAY ( G:GRAPH );
 var I,J:integer;
 begin
  for I:=1 to MAXNCDES do with G.NODES(.I.) do
   begin
    if PROCESSOR <> ROUTINE_0
     then
      begin
       writeln('NODE NUMBER ',I:2,':');
       writeln('     INPUT BUFFER:');
       for J:=1 tc CONNECTICNS do
        begin
          writeln('        PORT ',J:1,':');
          writeln('                DATUM = ',
                  INP(.J.).DATUM:8:5);
          case INP(.J.).SPECTRUM of
            BLACK: writeln('            ',
                           'WAVEFRONT IS BLACK');
            RED:   writeln('            ',
                           'WAVEFRONT IS RED');
            BLUE:  writeln('            ',
                           'WAVEFRONT IS BLUE');
            GREEN: writeln('            ',
                           'WAVEFRONT IS GREEN')
```

```
   end (* case *)
 end (* for *);
writeln('     CELL MEMORIES:');
for J:=1 tc CELL_MEMCRIES do
   writeln('        MEMORY ',J:1,':',
         MEM(.J.):8:5);
writeln('        OUTPUT BUFFER:');
for J:=1 tc CONNECTICNS do
   begin
     writeln('        PORT ',J:1,':');
     writeln('                DATUM = ',
            OUT(.J.).DATUM:8:5);
     case OUT(.J.).SPECTRUM of
       BLACK: writeln('              ',
                      'WAVEFRONT IS BLACK');
       RED:   writeln('              ',
                      'WAVEFRONT IS RED');
       BLUE:  writeln('              ',
                      'WAVEFRONT IS BLUE');
       GREEN: writeln('              ',
                      'WAVEFRONT IS GREEN')
     end (* case *)
    end (* for *);
if COLOR = (.WHITE.)
```

```
   then writeln('     COLOR IS WHITE')
else begin
      if CCIOR = (.YELLOW.) then
         writeln('     COLOR IS YELLOW');
      if CCIOR = (.RED.) then
         writeln('     COLOR IS RED');
      if CCIOR = (.BIUE.) then
         writeln('     COLOR IS BLUE');
      if CCICR = (.GREEN.) then
         writeln('     COLOR IS GREEN');
      if CCIOR = (.BLACK.) then
         writeln('     COLOR IS BLACK');
      if CCIOR = (.MAGENTA.) then
         writeln('     COLOR IS MAGENTA');
      if CCIOR = (.CYAN.) then
         writeln('     COLOR IS CYAN');
     end (* else-begin *);
case PROCESSOR of
  ROUTINE_1: writeln('     PROCESSOR IS GIVENS INT W/ SQRT');
  ROUTINE_2: writeln('     PROCESSOR IS GIVENS EXT W/ SQRT');
  ROUTINE_3: writeln('     PROCESSOR IS BUFFER CELL');
  ROUTINE_4: writeln('     PROCESSOR IS INNER PRODUCT STEP');
  ROUTINE_5: writeln('     PROCESSOR IS GIVENS INT W/C SQRT');
  ROUTINE_6: writeln('     PROCESSOR IS GIVENS EXT W/C SQRT');
```

```
    ROUTINE_7: writeln('     PROCESSOR IS BACK SUBSTITUTION');
    ROUTINE_8: writeln('     PROCESSOR IS DIVIDED DIFFERENCE');
  end (* case *);
  if TASK = IMPORTER
     then writeln('     TASK IS IMPORTER')
  else begin
        if TASK = INTERNAL then
          writeln('     TASK IS INTERNAL');
        if TASK = EXPORTER then
          writeln('     TASK IS EXPORTER')
       end (* else-begin *);
  case SHAPE cf
    AMORF:   writeln('     SHAPE IS AMORF');
    SQUARE:  writeln('     SHAPE IS SQUARE');
    OCTAGON: writeln('     SHAPE IS OCTAGON');
  end (* case *);
  writeln;
  writeln('     SCREEN COORDINATES:');
  writeln;
  writeln('             X = ',COORDX:1);
  writeln('             Y = ',COORDY:1);
  writeln;
  writeln;
end (* then begin *)
```

```
    end (* with *)
  end (* OUTPUT_ARRAY *);
procedure OUTPUT_TC_PRINTER ( G:GRAPH );
 var I,J:integer;
 begin
  for I:=1 to MAXNCDES do with G.NODES(.I.) do
   begin
    if PROCESSOR <> ROUTINE_0
     then
      begin
       writeln(REC,'NODE NUMBER ',I:2,':');
       writeln(REC,'     INPUT BUFFER:');
       for J:=1 tc CONNECTICNS do
        begin
          writeln(REC,'         PORT ',J:1,':');
          writeln(REC,'                DATUM = ',
                  INP(.J.).DATUM:8:5);
          case INP(.J.).SPECTRUM of
            BLACK: writeln(REC,'            ',
                           'WAVEFRONT IS BLACK');
            RED:   writeln(REC,'            ',
                           'WAVEFRONT IS RED');
            BLUE:  writeln(REC,'            ',
                           'WAVEFRONT IS BLUE');
```

```
      GREEN: writeln(REC,'            ',
                     'WAVEFRONT IS GREEN')
    end (* case *)
  end (* for *);
writeln(REC,'     CELL MEMORIES:');
for J:=1 to CELL_MEMORIES do
    writeln(REC,'         MEMORY ',J:1,':',
          MEM(.J.):8:5);
writeln(REC,'         OUTPUT BUFFER:');
for J:=1 to CONNECTIONS do
    begin
      writeln(REC,'         PORT ',J:1,':');
      writeln(REC,'                DATUM = ',
             OUT(.J.).DATUM:8:5);
      case OUT(.J.).SPECTRUM of
        BLACK: writeln(REC,'              ',
                       'WAVEFRONT IS BLACK');
        RED:   writeln(REC,'              ',
                       'WAVEFRONT IS RED');
        BLUE:  writeln(REC,'              ',
                       'WAVEFRONT IS BLUE');
        GREEN: writeln(REC,'              ',
                       'WAVEFRONT IS GREEN')
      end (* case *)
```

```
            end (* for *);
        if COLOR = (.WHITE.)
           then writeln(REC,'     COLOR IS WHITE')
        else begin
              if CCIOR = (.YELLOW.) then
                 writeln(REC,'     COLOR IS YELLOW');
              if CCIOR = (.RED.) then
                 writeln(REC,'     COLOR IS RED');
              if CCIOR = (.BIUE.) then
                 writeln(REC,'     COLOR IS BLUE');
              if CCICR = (.GREEN.) then
                 writeln(REC,'     COLOR IS GREEN');
              if CCIOR = (.BLACK.) then
                 writeln(REC,'     COLOR IS BLACK');
              if CCIOR = (.MAGENTA.) then
                 writeln(REC,'     COLOR IS MAGENTA');
              if CCIOR = (.CYAN.) then
                 writeln(REC,'     COLOR IS CYAN');
             end (* else-begin *);
      end (* then begin *)
   end (* with *)
 end (* OUTPUT_TO_PRINTER *);
prccedure OUTPUT_LINKS ( G:GRAPH; var LINKLIST:LINKTABIE );
(* output list of links and their assigned node/ports *)
```

```
var I,J,K,L,Y:integer;
begin
  Y:=1;
  writeln('LIST OF LINKS  ( CONNECTED NODE/PORTS ):');
  for I:=1 to MAXNCDES do
    for J:=1 to CONNECTIONS do
      for K:=1 to MAXNODES do
        for L:=1 to CONNECTIONS do
          begin
            if ADJACENT(G,I,K,J,L)
              then begin
                     writeln('     ORIGIN NODE ',I:2,' PORT ',
                             J:1,' / DESTINATION NODE ',K:2,
                             ' PORT ',L:1);
                     (* prepare work list of links *)
                     LINKLIST(.1,Y.):=I;
                     LINKLIST(.2,Y.):=J;
                     LINKLIST(.3,Y.):=K;
                     LINKLIST(.4,Y.):=L;
                     Y:=Y+1
                   end
          end;
end (* OUTPUT_LINKS *);
procedure COLOR_PROCESSING (var G:GRAPH; I:NODEPTR);
```

```
 (* mixes input primary colors as below:
     blue + red = magenta
     blue + green = cyan
     green + red = yellow
     blue + green + red = white
     black + any combination = that combination
     black + black = black
   and assigns result to COLOR of node I       *)
var CODE: array(.LEAD.) of RAINBOW;
    J:LEAD;
begin
  with G.NODES(.I.) do
   begin
    COLOR:=(. .);
    for J:=1 to CONNECTIONS do
      begin
       case INP(.J.).SPECTRUM of
         BLACK: CODE(.J.):=(.BLACK.);
         BLUE:  CODE(.J.):=(.BLUE.);
         RED:   CODE(.J.):=(.RED.);
         GREEN: CODE(.J.):=(.GREEN.);
       end (* case *);
       COLOR:=COLOR+CODE(.J.)
      end (* for *);
```

```
      if (.BLUE,GREEN,RED.) <= COLOR then COLOR:=(.WHITE.);
      if (.BLUE,GREEN.) <= COLOR     then COLOR:=(.CYAN.);
      if (.BLUE,RED.) <= COLOR       then COLOR:=(.MAGENTA.);
      if (.GREEN,RED.) <= COLOR      then COLOR:=(.YELLOW.);
      if (.RED.) <= COLOR            then COLOR:=(.RED.);
      if (.BLUE.) <= COLOR           then COLOR:=(.BLUE.);
      if (.GREEN.) <= COLOR          then COLOR:=(.GREEN.);
      if (.BLACK.) <= COLOR          then COLOR:=(.BLACK.)
    end (* with *)
  end (* COLOR_PROCESSING *);

(* library routines for cell processors *)

procedure GIVENS_INTERNAL;
  (* adopted nomenclature is:
      c(i,j-1) ==> INP(.1.)
      s(i,j-1) ==> INP(.2.)
      z(i,j)   ==> INP(.3.)
      c(i,j)   ==> OUT(.1.)
      s(i,j)   ==> OUT(.2.)
      z(i+1,j) ==> OUT(.3.)
      r(i,j)   ==> MEM(.1.)            *)
 begin
   with G.NODES(.I.) do
    begin
```

```
      OUT(.1.):=INP(.1.);
      OUT(.2.):=INP(.2.);
      OUT(.3.).DATUM:=INP(.1.).DATUM*INP(.3.).DATUM-
                      INP(.2.).DATUM*MEM(.1.);
      OUT(.3.).SPECTRUM:=INP(.3.).SPECTRUM;
      MEM(.1.):=INP(.1.).DATUM*MEM(.1.)+INP(.2.).DATUM*INP(.3.).DATUM;
      COLOR_PROCESSING(G,I)
    end (* with *)
 end (* GIVENS_INTERNAL *);
procedure GIVENS_EXTERNAL;
 (* adopted nomenclature is:
    z(i,j) ==> INP(.1.)
    c(i,j) ==> OUT(.1.)
    s(i,j) ==> OUT(.2.)
    r(i,j) ==> MEM(.1.)        *)
 begin
   with G.NODES(.I.) do
    begin
      if INP(.1.).DATUM = 0.0
        then begin
               OUT(.1.).DATUM:=1.0;
               OUT(.2.).DATUM:=0.0
             end
      else begin
```

```
                OUT(.1.).DATUM:=MEM(.1.)/SQRT(SQR(MEM(.1.))
                                + SQR(INP(.1.).DATUM));
                OUT(.2.).DATUM:=INP(.1.).DATUM/SQRT(SQR(MEM(.1.))
                                + SQR(INP(.1.).DATUM));
                MEM(.1.):=SQRT(SQR(MEM(.1.))+SQR(INP(.1.).LATUM))
              end (* else-begin *);
        OUT(.1.).SPECTRUM:=INP(.1.).SPECTRUM;
        OUT(.2.).SPECTRUM:=INP(.1.).SPECTRUM;
        COLOR_PROCESSING(G,I)
      end (*with *)
  end (* GIVENS_EXTERNAL *);
procedure BUFFER_CELL;
      begin
        with G.NODES(.I.) do begin
              OUT(.1.):=INP(.1.);
              OUT(.2.):=INP(.2.);
              OUT(.3.):=INP(.3.);
              MEM(.1.):=INP(.1.).DATUM;
              MEM(.2.):=INP(.2.).DATUM;
              MEM(.3.):=INP(.3.).DATUM;
              COLOR_PROCESSING (G,I);
        end (* with *);
      end (* BUFFER_CELL *);
procedure INNER_PRODUCT;
```

```
    (* nomenclature is
        A(i)  ==> INP(.1.)
        B(i)  ==> INP(.2.)
        C(i)  ==> INP(.3.)
        A(o)  ==> OUT(.1.)
        B(o)  ==> OUT(.2.)
        C(o)  ==> OUT(.3.)
        C(o)  ==> MEM(.1.)
      algorithm is  C(o) <== C(i) + A(i) * B(i)          *)
    begin
      with G.NODES(.I.) do begin
        OUT(.1.):=INP(.1.);
        OUT(.2.):=INP(.2.);
        MEM(.1.):=INP(.3.).DATUM+INP(.1.).DATUM*INP(.2.).DATUM;
        OUT(.3.).DATUM:=MEM(.1.);
        OUT(.3.).SPECTRUM:=INP(.3.).SPECTRUM;
        COLOR_PROCESSING (G,I);
      end (* with *);
    end (* INNER_PRODUCT *);
procedure GIVENS_INT_WO;
   (* nomenclature is
       X(i)  ==> INP(.1.)
       C(i)  ==> INP(.2.)
       S(i)  ==> INP(.3.)
```

```
        Z(i) ==> INP(.4.)
        X(o) ==> OUT(.1.)
        C(o) ==> OUT(.2.)
        S(o) ==> OUT(.3.)
        Z(o) ==> OUT(.4.)
        R    ==> MEM(.1.)
      algorithm is
        X(o)<--X(i)-Z(i)*R
        R   <--C(i)*R+S(i)*X(i)
    begin
      with G.NODES(.I.) do begin
        OUT(.1.).DATUM:=INP(.1.).DATUM-INP(.4.).DATUM*MEM(.1.);
        OUT(.1.).SPECTRUM:=INP(.1.).SPECTRUM;
        MEM(.1.):=INP(.2.).DATUM*MEM(.1.)+INP(.3.).DATUM*INP(.1.).DATUM;
        OUT(.2.):=INP(.2.);
        OUT(.3.).DATUM:=INP(.3.).DATUM;
        OUT(.4.).DATUM:=INP(.4.).DATUM;
        COLOR_PROCESSING (G,I);
      end (* with *);
    end (* GIVENS_INT_WO *);
procedure GIVENS_EXT_WO;
    (* nomenclature is
         DEL(i) ==> INP(.1.)
         X(i)   ==> INP(.2.)
```

```
      DEL(o) ==> OUT(.1.)
      C(o)   ==> OUT(.2.)
      S(o)   ==> OUT(.3.)
      Z(o)   ==> OUT(.4.)
      D      ==> MEM(.1.)
      D'     ==> MEM(.2.)
   algorithm is
      if X(i)=0 or DEL(i)=0 then
               C(o)   <--1
               S(o)   <--0
               Z(o)   <--X(i)
               DEL(o)<--DEL(i)
      else
               D'     <--D+DEL(i)*X(i)**2
               C(o)   <--D/D'
               S(o)   <--DEL(i)*X(i)/D'
               Z(o)   <--X(i)
               D      <--D'
               DEL(o)<--C(o)*DEL(i)
      endif                                                    *)
begin
   with G.NODES(.I.) do begin
     if ((INP(.2.).DATUM=0.0) or (INP(.1.).DATUM=0.0)) then
        begin
```

```
                OUT(.2.).DATUM:=1.0;
                OUT(.3.).DATUM:=0.0;
                OUT(.4.):=INP(.2.);
                OUT(.1.):=INP(.1.);
              end
            else begin
                MEM(.2.):=MEM(.1.)+INP(.1.).DATUM*INP(.2.).DATUM*
                          INP(.2.).DATUM;
                OUT(.2.).DATUM:=MEM(.1.)/MEM(.2.);
                OUT(.3.).DATUM:=INP(.1.).DATUM*INP(.2.).DATUM/MEM(.2.);
                OUT(.4.):=INP(.2.);
                MEM(.1.):=MEM(.2.);
                OUT(.1.).DATUM:=OUT(.2.).DATUM*INP(.1.).DATUM;
                OUT(.1.).SPECTRUM:=INP(.1.).SPECTRUM;
             end (* else *);
             COLOR_PROCESSING (G,I);
         end (* with *);
    end (* GIVENS_EXT_WO *);
procedure BACK_SUBSTITUTION;
  (* nomenclature is
       A(i,i) ==> INP(.1.)
       B(i)   ==> INP(.2.)
       Y(i)   ==> INP(.3.)
       X(i)   ==> OUT(.1.)
```

```
        X(i)    ==> MEM(.1.)
     algorithm is
        X(i) <-- (. B(i)-Y(i) .) / A(i,i)
    begin
       with G.NODES(.I.) do begin
         if ( INP(.1.).DATUM <> 0.0 ) then
            MEM(.1.):=( INP(.2.).DATUM-INP(.3.).DATUM )/INP(.1.).DATUM
         else MEM(.1.):=0.0;
         OUT(.1.).DATUM:=MEM(.1.);
         OUT(.1.).SPECTRUM:=INP(.2.).SPECTRUM;
         COLOR_PROCESSING (G,I);
       end (* with *);
    end (* BACK_SUBSTITUTION *);
procedure DIVIDED_DIFFERENCE;
  (* nomenclature is
       X(i)      ==> INP(.1.)
       X(i+m)    ==> INP(.2.)
       Y(i,j)    ==> INP(.3.)
       Y(i+1,j)  ==> INP(.4.)
       X(i+m)    ==> OUT(.1.)
       X(i)      ==> OUT(.2.)
       Y(i,j+1)  ==> OUT(.3.)
       Y(i,j+1)  ==> OUT(.4.)
     algorithm is
```

```
      Y(i,j+1) <-- (.Y(i+1,j)-Y(i,j).)/(.X(i+m)-X(i).)
    begin
      with G.NODES(.I.) do begin
       OUT(.2.):=INP(.1.);
       OUT(.1.):=INP(.2.);
       if (abs(INP(.2.).DATUM - INP(.1.).DATUM) < 0.0000001)
        then writeln('DELTA X INPUT IS TOO SMALL AT NODE ',I:2,
                     '; PREVIOUS RESULT IS KEPT')
       else MEM(.1.):=(INP(.4.).DATUM-INP(.3.).DATUM)/
                      (INP(.2.).DATUM-INP(.1.).DATUM);
       OUT(.3.).DATUM:=MEM(.1.);
       OUT(.4.).DATUM:=MEM(.1.);
       OUT(.3.).SPECTRUM:=OUT(.1.).SPECTRUM;
       OUT(.4.).SPECTRUM:=OUT(.2.).SPECTRUM;
       COLOR_PROCESSING (G,I);
      end (* with *);
    end (* DIVIDED_DIFFERENCE *);

(* routines to display array on screen *)

procedure REFRESH_SCREEN ( G:GRAPH );
    var COLORCODE:1..8;
        I:integer;
    begin
      for I:=1 to MAXNODES do with G.NODES(.I.) do
```

```
        if ( PROCESSOR <> ROUTINE_0 ) then
         begin
           if (COLOR = (.BLACK.)) then COLORCODE:=1
           else if (COLOR = (.RED.)) then COLORCODE:=2
           else if (COLOR = (.GREEN.)) then COLORCODE:=3
           else if (COLOR = (.BLUE.)) then COLORCODE:=4
           else if (COLOR = (.CYAN.)) then COLORCODE:=5
           else if (COLOR = (.MAGENTA.)) then COLORCODE:=6
           else if (COLOR = (.YELLOW.)) then COLORCODE:=7
           else if (COLOR = (.WHITE.)) then COLORCODE:=8;
           case SHAPE of
             SQUARE: SQUARED (COLORCODE,COORDX,COORDY);
             OCTAGON:OCTGON (COLORCODE,COORDX,COORDY);
           end (* case SHAPE *);
           FILCEL (COORDX,COORDY,MEM(.1.));
         end (* if-then *)
    end (* REFRESH_SCREEN *);
procedure INITIALIZE_SCREEN ( G:GRAPH; LINKLIST:LINKTABLE );
    var COLORCODE:1..8;
        I,POINTING:integer;
    begin
      COLORCODE:=1; (* black *)
      for I:=1 to MAXNODES do with G.NODES(.I.) do
        if ( PROCESSOR <> ROUTINE_0 ) then
```

```
            begin
              case SHAPE of
                SQUARE: SQUARED (COLORCODE,COORDX,COORDY);
                OCTAGON:OCTGON (COLORCODE,COORDX,COORDY);
              end (* case SHAPE *);
              FILCEI (COORDX,COORDY,MEM(.1.));
              POINTING:=I;
              IDENT (COORDX,CCORDY,POINTING);
            end (* if-then *);
        for I:=1 to IINKNUMB do
              ARROW (G.NODES(.LINKLIST(.1,I.).).COORDX,
                     G.NODES(.LINKLIST(.1,I.).).COORDY,
                     G.NODES(.LINKLIST(.3,I.).).COORDX,
                     G.NODES(.LINKLIST(.3,I.).).COORDY);
     end (* INITIAIIZE_SCREEN *);
procedure CORRECT_NODE ( var G:GRAPH );
   var NODEID,NEWID:NODEPTR;
       CHANGECODE:char;
       NEWCODE:integer;
   begin
     writeln('ENTER ID NUMBER OF WRONG NODE');
     readln(NODEID);
     with G.NODES(.NODEID.) do
      begin
```

```
writeln('SELECT ITEM CHANGE CODE FROM MENU:');
writeln;
writeln('       ID NUMBER    ==> 1');
writeln('       PROCESSOR    ==> 2');
writeln('       TASK         ==> 3');
writeln('       ABCISSA (X)  ==> 4');
writeln('       ORDINATE (Y) ==> 5');
writeln('       SHAPE        ==> 6');
read(CHANGECODE);
readln;
case CHANGECODE of
   '1': begin
          writeln('ENTER NEW ID NUMBER');
          readln(NEWID);
          G.NODES(.NEWID.).INP:=INP;
          G.NODES(.NEWID.).MEM:=MEM;
          G.NODES(.NEWID.).OUT:=OUT;
          G.NODES(.NEWID.).COLOR:=COLOR;
          G.NODES(.NEWID.).PROCESSOR:=PROCESSOR;
          G.NODES(.NEWID.).TASK:=TASK;
          G.NODES(.NEWID.).COORDX:=COORDX;
          G.NODES(.NEWID.).COORDY:=COORDY;
          G.NODES(.NEWID.).SHAPE:=SHAPE;
          PROCESSOR:=ROUTINE_0;
```

```
        COLOR:=(.BLACK.);
        TASK:=INTERNAL;
        CCORDX:=1;
        CCORDY:=1;
        SHAPE:=AMORF;
        writeln('OK, CHANGE IS DONE');
      end (* '1' *);
'2': begin
       writeln('SELECT NEW PROCESSOR CODE');
       writeln;
       writeln('   GIVENS INTERNAL W/  SQRT ==> 1');
       writeln('   GIVENS EXTERNAL W/  SQRT ==> 2');
       writeln('   BUFFER CELL              ==> 3');
       writeln('   INNER PRODUCT STEP       ==> 4');
       writeln('   GIVENS INTERNAL W/O SQRT ==> 5');
       writeln('   GIVENS EXTERNAL W/O SQRT ==> 6');
       writeln('   BACK SUBSTITUTION        ==> 7');
       writeln('   DIVIDED DIFFERENCE       ==> 8');
       readln(NEWCODE);
       case NEWCODE of
          1: PROCESSOR:=ROUTINE_1;
          2: PROCESSOR:=ROUTINE_2;
          3: PROCESSOR:=ROUTINE_3;
          4: PROCESSOR:=ROUTINE_4;
```

```
                5: PROCESSOR:=ROUTINE_5;
                6: PROCESSOR:=ROUTINE_6;
                7: PROCESSOR:=ROUTINE_7;
                8: PROCESSOR:=ROUTINE_8;
            end (*case *);
            writeln('OK, CHANGE IS DONE');
            end (* '2' *);
        '3': begin
            writeln('SELECT NEW TASK CODE');
            writeln;
            writeln('    IMPORTER ==> 1');
            writeln('    INTERNAL ==> 2');
            writeln('    EXPORTER ==> 3');
            readln(NEWCODE);
            case NEWCODE of
                1: TASK:=IMPORTER;
                2: TASK:=INTERNAL;
                3: TASK:=EXPORTER;
            end (* case *);
            writeln('OK, CHANGE IS DONE');
          end (* '3' *);
        '4': begin
            writeln('ENTER NEW ABCISSA');
            readln(NEWCODE);
```

```
                COORDX:=NEWCODE;
                writeln('OK, CHANGE IS DONE');
              end (* '4' *);
          '5': begin
                writeln('ENTER NEW ORDINATE');
                readln(NEWCODE);
                COORDY:=NEWCODE;
                writeln('OK, CHANGE IS DONE');
              end (* '5' *);
          '6': begin
                writeln('SELECT NEW SHAPE CODE');
                writeln;
                writeln('     SQUARE  ==> 1');
                writeln('     OCTAGON ==> 2');
                readln(NEWCODE);
                case NEWCODE of
                  1: SHAPE:=SQUARE;
                  2: SHAPE:=OCTAGON;
                end (* case *);
                writeln('OK, CHANGE IS DONE');
              end (* '6' *);
      end (* case CHANGECODE *);
    end (* with *);
end (* CORRECT_NODE *);
```

```
procedure CORRECT_LINK ( var G:GRAPH );
   var OPCODE:0..9;
       OK,LINK_IS_FIXED:boolean;
       CORRECT_CODE:char;
   begin
     LINK_IS_FIXED:=false;
     while not LINK_IS_FIXED do
      begin
         writeln('YOU WILL BE ALLOWED TO INSERT A NEW LINK OR');
         writeln('TO REMOVE AN EXISTING LINK;  IF  YOUR  NEED');
         writeln('REQUIRES BOTH OPERATIONS, DO ONE AT A TIME.');
         writeln;
         writeln('ENTER ID NUMBER OF ORIGIN CELL');
         readln(CELLORIG);
         writeln('ENTER OUTPUT PORT NUMBER FROM ORIGIN CELL');
         readln(OUTPORT);
         writeln('ENTER ID NUMBER OF DESTINATION CELL');
         readln(CELLDEST);
         writeln('ENTER INPUT PORT NUMBER AT DESTINATION CELL');
         readln(INPORT);
         OK:=false;
         while not OK do
           begin
             writeln('SELECT YOUR OPERATION CODE:');
```

```
    writeln;
    writeln('      REMOVE LINK ==> 1');
    writeln('      INSERT LINK ==> 2');
    readln(OPCODE);
    if (OPCODE = 1) then
       begin
         REMOVE (G,CELLORIG,CELLDEST,OUTPORT,INPORT);
         CK:=true;
       end
    else if (OPCODE = 2) then
       begin
         JOIN (G,CELLORIG,CELLDEST,OUTPORT,INPORT);
         CK:=true;
       end
    else writeln('TRY AGAIN');
end (* while *);
OK:=false;
while not OK do
 begin
  writeln('ANY OTHER LINK FIXING ? ENTER ''Y'' OR ''N''');
  read(CORRECT_CODE);
  readln;
  case CORRECT_CODE of
    'y','Y': OK:=true (* but loop again *);
```

```
                'n','N':
                          begin
                            OK:=true;
                            LINK_IS_FIXED:=true;
                          end;
               otherwise writeln('TRY AGAIN');
              end (* case *);
             end (* while *);
        end (* while *);
   end (* CORRECT_IINK *);
begin
(* initialize graph and graphics device *)
INITIO;
CLRSCR;
INITIALIZE_GRAPH (G);
(* selection of type of job; if old job is selected, take measures
  to initialize the screen before passing control to speedy
  review control block.                                          *)
writeln;
writeln;
writeln('DO YOU WANT TO START A NEW JOB OR TO RUN A PREVIOUS ONE ?');
writeln;
writeln('ENTER ''N'' FOR NEW OR ''P'' FOR PREVIOUS');
read(ECHOCODE);
```

```
readln;
writeln('DO YOU WANT TO BE ABLE TO FREEZE AUTOMATIC GRAPHICAL');
writeln('REVIEW BETWEEN CLOCK CYCLES ? ''Y'' OR ''N'' ?');
read(INTERRUPT_CLOCK);
readln;
writeln('DO YOU WANT TO RECORD THE SESSION FOR LATER PRINTING ?');
writeln('          ENTER ''Y'' OR ''N''');
read(RECCODE);
readln;
if (( RECCODE = 'Y') or ( RECCODE = 'y')) then
  begin
    open(REC,'SYSPRINT.DOC');
    rewrite(REC);
    WANT_REC:=true;
  end
else WANT_REC:=false;
if (( ECHOCODE = 'P' ) or ( ECHOCODE = 'p' ))
     then
      begin
        writeln;
        writeln('JUST A MOMENT');
        (* open file from previous job for reading *)
        open (CELIS,'SYSARRAY.MEM',history:=old);
        reset (CELIS);
```

```
            (* recalculate links to initialize screen *)
            G:=CELLS@;
            Y:=1;
            for I:=1 tc MAXNODES do
             for J:=1 to CONNECTIONS do
              for K:=1 to MAXNODES do
               for L:=1 to CONNECTIONS do
                 if ADJACENT (G,I,K,J,L) then
                   begin
                    LINKLIST(.1,Y.):=I;
                    LINKLIST(.2,Y.):=J;
                    LINKLIST(.3,Y.):=K;
                    LINKLIST(.4,Y.):=L;
                    Y:=Y+1;
                   end;
            LINKNUMB:=Y-1;
            INITIALIZE_SCREEN (G,LINKLIST);
            get(CELLS); (* to get in phase with sequence at
                                              statement 100 *)
            ECHOCODE:='Y';  (* because of statement 100 check *)
            goto 100 (* pass control to speedy review block *)
          end
else (* start new job *);
(* open processing memory file *)
```

```
open (CELLS,'SYSARRAY.MEM');
rewrite (CELLS);
(* define nodes *)
DEFINE_NODES (G);
(* define arcs *)
DEFINE_ARCS (G);
(* echo user defined graph *)
(* take space *)
for M:=1 to 5 do writeln;
(* output list of user defined nodes and their assignments and
  take measures to correct any wrong node assignment.      *)
writeln('LIST OF WORKING NODES AND THEIR DEFINITION BY THE USER:');
writeln;
OUTPUT_ARRAY (G);
NODE_IS_WRONG:=true;
writeln;
while NODE_IS_WRONG do
 begin
   writeln('ANYTHING TO CHANGE IN A NODE ? ENTER ''Y'' OR ''N''');
   read(CORRECTION_CODE);
   readln;
   if (( CORRECTION_CODE = 'Y' ) or ( CORRECTION_CODE = 'y')) then
      begin
        CORRECT_NODE (G);
```

```
        writeln('DO YOU WANT TO SEE RESULT OF CORRECTION ? ',
                'ENTER ''Y'' OR ''N''');
        read(CORRECTION_CODE);
        readln;
        if ((CORRECTION_CODE = 'Y') or (CORRECTION_CODE = 'y'))
            then OUTPUT_ARRAY (G)
        else;
      end (* if-then *)
   else if (( CORRECTION_CODE = 'N' ) or ( CORRECTION_CODE = 'n'))
                    then NODE_IS_WRONG:=false
   else writeln ('TRY AGAIN; ATTENTION ON TYPING !');
 end (* while *);
(* output list of links and their assigned node/ports and take
  measures to correct any wrong link connection.              *)
OUTPUT_LINKS (G,LINKLIST);
LINK_IS_WRONG:=true;
writeln;
while LINK_IS_WRONG do
 begin
   writeln('ANYTHING TO CHANGE IN A LINK ? ENTER ''Y'' OR ''N''');
   read(CORRECTION_CODE);
   readln;
   if ((CORRECTION_CODE = 'Y') or (CORRECTION_CODE = 'y'))
       then begin
```

```
                CORRECT_LINK (G);
                OUTPUT_LINKS (G,LINKLIST);
              end (* if-then *)
    else if ((CORRECTION_CODE = 'N') or (CORRECTION_CODE = 'n'))
          then LINK_IS_WRONG:=false
    else writeln('TRY AGAIN; WATCH ON TYPING');
  end (* while *);
(* snapshot of systolic array *)
CELLS@:=G;
put (CELLS);
(* initialize graphics device screen *)
INITIALIZE_SCREEN (G,LINKLIST);
(* process data till last clock cycle *)
  (* start of systclic processing *)
  writeln;
  X:=1; (* initialize systolic control variable *)
  writeln('ENTER NUMBER OF CLOCK CYCLES TO EXECUTE');
  readln(TIMER);
  while X <= TIMER do begin
    (* pulsation of cells *)
    for Y:=1 to LINKNUMB do (* transfer output buffers to other
                                                        cells *)
      G.NODES(.LINKLIST(.3,Y.).).INP(.LINKLIST(.4,Y.).):=
          G.NODES(.LINKLIST(.1,Y.).).OUT(.LINKLIST(.2,Y.).);
```

```
for I:=1 to MAXNODES do
  (* input to importers and internal processing of each cell *)
  with G.NODES(.I.) do
   begin
     case TASK of
      IMPORTER:      (* input to importers *)
       begin
        writeln('NOW YOU''LL ENTER INPUTS TO CELI NUMBER ',
                I:2,' AT TIME t=',X:2);
        INPUT_CHECK:='N';
        repeat
           writeln('HOW MANY EXTERNAL INPUT PORTS ARE ',
                   'ACTIVE?');
           readln(INPUTKEY);
           writeln(INPUTKEY:2,' EXTERNAL INPUT PORTS ? ''Y''',
                   ' OR ''N'' ?');
           read(INPUT_CHECK);
           readln;
        until (( INPUT_CHECK = 'Y' ) or ( INPUT_CHECK = 'y' ));
        INPUT_CHECK:='N';
        for L:=1 to INPUTKEY do begin
          repeat
             writeln('ENTER ID NUMBER OF PORT ',I:1);
             readln(INPORT);
```

```
      writeln('PORT ID IS ',INPORT:2,' ? ''Y'' OR ''N'' ?');
      read(INPUT_CHECK);
      readln;
   until ((INPUT_CHECK='Y') or (INPUT_CHECK='y'));
   INPUT_CHECK:='N';
   repeat
      writeln('ENTER INPUT DATUM TO PORT ID NUMBER ',
              INPORT:1);
      readln(INP(.INPORT.).DATUM);
      writeln('INPUT DATUM IS ',INP(.INPORT.).DATUM:9:3,
              ' ? ''Y'' OR ''N'' ?');
      read(INPUT_CHECK);
      readln;
   until ((INPUT_CHECK='Y') or (INPUT_CHECK='y'));
   INPUT_CHECK:='N';
   repeat
      writeln('ENTER COLOR CODE FOR THIS DATUM ',
              'WAVEFRONT :');
      writeln('         BLACK ==> 1');
      writeln('         BLUE  ==> 2');
      writeln('         RED   ==> 3');
      writeln('         GREEN ==> 4');
      readln(COLORNUMBER);
      case COLORNUMBER of
```

```
                1: INP(.INPORT.).SPECTRUM:=BLACK;
                2: INP(.INPORT.).SPECTRUM:=BLUE;
                3: INP(.INPORT.).SPECTRUM:=RED;
                4: INP(.INPORT.).SPECTRUM:=GREEN
              end (*case *);
              writeln('WAVEFRONT COLORCODE IS ',COLORNUMBER:1,
                      ' ? ''Y'' OR ''N'' ?');
              read(INPUT_CHECK);
              readln;
            until ((INPUT_CHECK='Y') or (INPUT_CHECK='y'));
            INPUT_CHECK:='N';
        end (* for *)
          end (* IMPORTER *);
         EXPORTER: (* do nothing *);
         INTERNAL: (* do nothing *)
        end (* case *);
        (* processing of data into cell *)
        case PROCESSOR of
          ROUTINE_0: (* do nothing *);
          ROUTINE_1: GIVENS_INTERNAL;
          ROUTINE_2: GIVENS_EXTERNAL;
          ROUTINE_3: BUFFER_CELL;
          ROUTINE_4: INNER_PRODUCT;
          ROUTINE_5: GIVENS_INT_WO;
```

```
            ROUTINE_6: GIVENS_EXT_WO;
            ROUTINE_7: BACK_SUBSTITUTION;
            ROUTINE_8: DIVIDED_DIFFERENCE;
          end (* case *)
       end (* for-with *);
     X:=X+1;
     (* output list of cells and their status *)
     writeln('LIST OF CELLS AND THEIR TRANSITORY STATUS:');
     writeln;
     if ( WANT_REC = true ) then
       begin
        writeln(REC);
        writeln(REC);
        writeln(REC,'STATUS OF CELLS AT TIME t = ',(x-1):2);
        writeln(REC);
        OUTPUT_TO_PRINTER (G);
       end;
     OUTPUT_ARRAY (G);
     REFRESH_SCREEN (G);
     CELLS@:=G;
     put (CELLS);
   end (* while *);
   WANT_REC:=false; (* to avoid risk of recording twice *)
(* speedy review of whole sequence *)
```

```
    writeln('DO YOU WANT TO REVIEW THE WHOLE SEQUENCE ? ENTER ',
            '''Y'' OR ''N''');
    read(ECHOCODE);
    readln;
    reset (CELLS);
100:if (( ECHOCODE = 'Y') or ( ECHOCODE = 'y' )) then
     begin
       ERRORCODE:=status (CELLS);
       if ( ERRORCODE > 0 )
           then writeln('FILE ACCESS ERROR NUMBER ',ERRORCODE:3)
       else if ( ERRORCODE < 0 )
           then writeln('FILE ''SYSARRAY.MEM'' IS EMPTY')
       else writeln('START OF PRESENTATION');
       X:=1;
       while not eof (CELLS) do
         begin
           G:=CELLS@;
           if ( WANT_REC = true ) then
              begin
                writeln(REC);
                writeln(REC);
                writeln(REC,'STATUS OF CELLS AT TIME t = ',X:2);
                writeln(REC);
                OUTPUT_TO_PRINTER (G);
```

```
            end
          else;
          GO_AHEAD:='N';
          if ((INTERRUPT_CLOCK='Y') or (INTERRUPT_CLOCK='y')) then
            repeat
              writeln('START NEXT CLOCK CYCLE ? ''Y'' OR ''N'' ?');
              read(GC_AHEAD);
              readln;
            until ((GO_AHEAD='Y') or (GO_AHEAD='y'))
          else;
          X:=X+1;
          REFRESH_SCREEN (G);
          get (CELLS);
        end (* while *);
    end (* if-then *)
  else (* terminate *);
  close (CELLS);
writeln('LOOK AT YOUR  DIRECTORY  FOR  FILE  SYSARRAY.MEM; IT');
writeln('CONTAINS A RECORDING OF THE PROCESSING FOR  GRAPHICS');
writeln('REVIEW; IF YOU ALSO ASKED FOR RECORDING FOR PRINTING');
writeln('THERE IS ANOTHER RECORD CALLED SYSPRINT.DOC THAT YOU');
writeln('CAN ASK THE OPERATING SYSTEM TO SEND TO  PRINTER  IF');
writeln('YOU WISH;');
writeln;
```

```
 writeln('        THIS IS THE END OF THE SESSION ');
 (* finish with graphics device *)
 FINIT;
end (* SYSTOLIC_SIMULATION *).
```

## APPENDIX B

## SUBROUTINES FOR IMPLEMENTING GRAPHICS PRIMITIVES

Note: These subroutines are written in FORTRAN 77 and are able to be compiled by the VAX FORTRAN Compiler. They call subroutines available at the SIGGRAPH CORE GRAPHICS PACKAGE from George Washington University. They have been set up for interfacing the RAMTEK Graphics Device.

```
SUBROUTINE INITIO     ! Initializes graphics operation
  call INIT (2,0,2,0)
  call NITSRF (3,1,2)
  call SELSRF (3)
  call SCLIPW(.TRUE.)
  CALL SCORTP(1)
  CALL SCHPRE(1)
  call SLWID (0.2)
  CALL SCHSIZ (5.0,7.0)
  call DCLNDX (3,1,0.0,0.0,0.0)
  call DCLNDX (3,2,0.8,0.0,0.0)
  call DCLNDX (3,3,0.0,0.8,0.0)
  call DCLNDX (3,4,0.0,0.0,0.7)
  call DCLNDX (3,5,0.0,0.8,0.8)
```

```
  call DCLNDX (3,6,0.8,0.0,0.8)
  call DCLNDX (3,7,0.8,0.8,0.0)
  call DCLNDX (3,8,1.0,1.0,1.0)
  call SWINDO(0.0,400.0,0.0,400.0)
  call CRRSEG (1)
  call SBGNDX (1)
RETURN
END

SUBROUTINE FINIT   ! Terminates graphics operation
  call CLRSEG
  call DELSRF (3)
  call TRMSRF (3)
  call TERM
  call EXIT
RETURN
END

SUBROUTINE CLRSCR  ! Establishes white background
  real*4 x(4),y(4)
  x(1)=0.0
  y(1)=0.0
  x(2)=400.0
  y(2)=0.0
  x(3)=400.0
```

```
      y(3)=400.0
      x(4)=0.0
      y(4)=400.0
      call SPISTY (2)
      call SFNDX (8)
      call SLNDX (1)
      call POLYA2 (x,y,4)
    RETURN
    END

    SUBROUTINE SQUARED (color,coordx,coordy) ! Draws a square given
   c                                          color and center position
      integer*2 color,coordx,coordy
      real*4 xc,yc,x(4),y(4)
      xc=50.0*coordx-25.0
      yc=50.0*coordy-25.0
      x(1)=xc-20.0
      x(2)=xc+20.0
      x(3)=x(2)
      x(4)=x(1)
      y(1)=yc-18.0
      y(2)=y(1)
      y(3)=yc+18.0
      y(4)=y(3)
      call SPISTY (2)
```

```
      call SFNDX (color)
      call SLNDX (1)
      call POLYA2 (x,y,4)
    RETURN
    END

  SUBROUTINE OCTGON (color,coordx,coordy)  ! Draws a octagon given
c                                                color and central position
    integer*2 color,coordx,coordy
    real*4 xc,yc,x(8),y(8)
    xc=50.0*coordx-25.0
    yc=50.0*coordy-25.0
    x(1)=XC-8.0
    y(1)=YC-20.0
    x(2)=XC+8.0
    y(2)=YC-20.0
    x(3)=XC+20.0
    y(3)=YC-8.0
    x(4)=XC+20.0
    y(4)=YC+8.0
    x(5)=XC+8.0
    y(5)=YC+20.0
    x(6)=XC-8.0
    y(6)=YC+20.0
    x(7)=XC-20.0
```

```
      y(7)=YC+8.0
      x(8)=XC-20.0
      y(8)=YC-8.0
      call SPISTY (2)
      call SFNDX (cclor)
      call SLNDX (1)
      call POLYA2 (x,y,8)
    RETURN
    END

    SUBROUTINE FILCEL (coordx,coordy,numb) ! Places a number
c                              into a cell whose coordinates are given
      integer*2 coordx,coordy,ti,i,p(7)
      real*4 xc,yc,numb,xo,yc,numb1,tr
      xc=50.0*coordx-25.0
      yc=50.0*coordy-25.0
      xo=xc-20.0
      yo=yc-3.0
      call MOVA2 (xc,yo)
      call STNDX (8)
      if ((numb .gt. 9999.999) .or. (numb .lt. -9999.999)) then
       write (6,100)
100    format('  CONTENTS OF A CELL ARE TOO LARGE; DISREGARD RESULTS')
       write (6,200) coordx,coordy
200    format('  CELL COORDINATES ARE X=',i2,' AND Y=',i2)
```

```
      endif
      if (numb .lt. 0.0) then
         call TEXT ('-',1)
         call MOVR2 (4.0,0.0)
      else
         call TEXT ('+',1)
         call MOVR2 (4.0,0.0)
      endif
      numb1=abs(numb)
      numb1=numb1/10000.0
      do 10 i=1,7
         tr=numb1*10.0
         ti=ifix(tr)
         p(i)=ti
         numb1=tr-float(ti)
10    continue
      do 20 i=1,7
         if (i .eq. 5) then
            call TEXT ('.',1)
            call MOVR2 (4.3,0.0)
         endif
         if (p(i) .eq. 1) then
            call TEXT ('1',1)
         else if (p(i) .eq. 2) then
```

```
            call TEXT ('2',1)
          else if (p(i) .eq. 3) then
            call TEXT ('3',1)
          else if (p(i) .eq. 4) then
            call TEXT ('4',1)
          else if (p(i) .eq. 5) then
            call TEXT ('5',1)
          else if (p(i) .eq. 6) then
            call TEXT ('6',1)
          else if (p(i) .eq. 7) then
            call TEXT ('7',1)
          else if (p(i) .eq. 8) then
            call TEXT ('8',1)
          else if (p(i) .eq. 9) then
            call TEXT ('9',1)
          else if (p(i) .eq. 0) then
            call TEXT ('0',1)
          endif
          call MOVR2 (4.3,0.0)
20      continue
        RETURN
        END

      SUBROUTINE IDENT (coordx,coordy,id)! print
c           identification label on cell bottom right
```

```
integer*2 coordx,coordy,id,p(2),id1,i
real*4 xo,yo,xc,yc
xc=50.0*coordx-25.0
yc=50.0*coordy-25.0
xo=xc+12.0
yo=yc-25.0
call MOVA2 (xc,yo)
call STNDX (8)
id1=id
id1=id1/10
p(1)=id1
p(2)=id-id1*10
do 10 i=1,2
   if (p(i) .eq. 1) then
      call TEXT ('1',1)
   else if (p(i) .eq. 2) then
      call TEXT ('2',1)
   else if (p(i) .eq. 3) then
      call TEXT ('3',1)
   else if (p(i) .eq. 4) then
      call TEXT ('4',1)
   else if (p(i) .eq. 5) then
      call TEXT ('5',1)
   else if (p(i) .eq. 6) then
```

```
            call TEXT ('6',1)
          else if (p(i) .eq. 7) then
             call TEXT ('7',1)
          else if (p(i) .eq. 8) then
             call TEXT ('8',1)
          else if (p(i) .eq. 9) then
             call TEXT ('9',1)
          else if (p(i) .eq. 0) then
             call TEXT ('0',1)
          endif
          call MOVR2 (4.5,0.0)
10      continue
        RETURN
        END

      SUBROUTINE ARROW (x1,y1,x2,y2)  ! Draws an arrow from a cell
c                                     to another given the coordinates of both
        integer*2 x1,y1,x2,y2
        real*4 xs,ys,xf,yf,xc,yc,alfa,lx,ly,alfa1,alf2
        real*4 x1a,y1a,x2a,y2a,pi,xl,yl,xr,yr
        pi=3.141593
        x1a=50.0*x1-25.0
        x2a=50.0*x2-25.0
        y1a=50.0*y1-25.0
```

```
y2a=50.0*y2-25.0
xc=(x1a+x2a)/2.0
yc=(y1a+y2a)/2.0
if ((x2 .eq. x1) .and. (y2 .gt. y1)) then
   alfa=pi/2.0
else if ((x2 .eq. x1) .and. (y2 .lt. y1)) then
   alfa=-pi/2.0
else if ((x2 .gt. x1) .and. (y2 .eq. y1)) then
   alfa=0.0
else if ((x2 .lt. x1) .and. (y2 .eq. y1)) then
   alfa=pi
else if ((x2 .gt. x1) .and. (y2 .gt. y1)) then
   alfa=atan2(y2a-y1a,x2a-x1a)
else if ((x2 .gt. x1) .and. (y2 .lt. y1)) then
   alfa=-atan2(-(y2a-y1a),x2a-x1a)
else if ((x2 .lt. x1) .and. (y2 .lt. y1)) then
   alfa=atan2(-(y2a-y1a),-(x2a-x1a)) - pi
else if ((x2 .lt. x1) .and. (y2 .gt .y1)) then
   alfa=atan2(y2a-y1a,x2a-x1a)
endif
lx=9.0*cos(alfa)
ly=9.0*sin(alfa)
xs=xc-lx/2.0
xf=xc+lx/2.0
```

```
ys=yc-ly/2.0
yf=yc+ly/2.0
call SLNDX (2)       ! make = 1 for black arrow
call MOVA2 (xs,ys)
call LINA2 (xf,yf)
alfa1=alfa+pi/6.0
alfa2=alfa-PI/6.0
xr=xf-5.0*cos(alfa1)
xl=xf-5.0*cos(alfa2)
yr=yf-5.0*sin(alfa1)
yl=yf-5.0*sin(alfa2)
call LINA2 (xr,yr)
call MOVA2 (xf,yf)
call LINA2 (xl,yl)
RETURN
END
```

# APPENDIX C

## A NUMERICAL EXAMPLE FOR MATRIX TRIANGULARIZATION

Suppose we have a system represented by the matrix equation AX = B and we want to solve it by Givens Rotations:

$$\begin{bmatrix} 2 & 4 & 1 \\ 5 & 7 & 4 \\ 3 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x(1) \\ x(2) \\ x(3) \end{bmatrix} = \begin{bmatrix} 12 \\ 3 \\ 8 \end{bmatrix}$$

The first elementary transformation matrix is

$$\begin{bmatrix} c & s & 0 \\ -s & c & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

where $d = \text{sqrt}(2x2+5x5) = 5.38516$

$c = 2/d = 0.37139$

$s = 5/d = 0.92848$

and so we will have

$$\begin{bmatrix} 0.37139 & 0.92848 & 0 \\ -0.92848 & 0.37139 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 4 & 1 \\ 5 & 7 & 4 \\ 3 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 5.38518 & 7.98492 & 4.08531 \\ -0.00001 & -1.11419 & 0.55708 \\ 3 & 0 & 1 \end{bmatrix}$$

The second elementary transformation matrix is

$$\begin{bmatrix} c & 0 & s \\ 0 & 1 & 0 \\ -s & 0 & c \end{bmatrix} \text{ where } d = \text{sqrt}(5.38518 \times 5.38518 + 3 \times 3) = 6.16443$$

$$c = 5.38518/d = 0.87359$$

$$s = 3/d = 0.48666$$

and sc we will have

$$\begin{bmatrix} 0.87359 & 0 & 0.48666 \\ 0 & 1 & 0 \\ -0.48666 & 0 & 0.87359 \end{bmatrix} \cdot \begin{bmatrix} 5.38518 & 7.98492 & 4.08531 \\ -0.00001 & -1.11419 & 0.55708 \\ 3 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 6.16442 & 6.97555 & 4.05555 \\ -0.00001 & -1.11419 & 0.55708 \\ 0.00002 & -3.88594 & -1.11457 \end{bmatrix}$$



The third elementary transformation matrix is

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & c & s \\ 0 & -s & c \end{bmatrix} \text{ where } d = \text{sqrt}((-1.11419) \times (-1.11419) + (-3.88594) \times (-3.88594)) = 4.04252$$

$$c = -1.11419/d = -0.27562$$

$$s = -3.88594/d = -0.96127$$

and so we will have

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -0.27562 & -0.96127 \\ 0 & 0.96127 & -0.27562 \end{bmatrix} \cdot \begin{bmatrix} 6.16442 & 6.97555 & 4.05555 \\ -0.00001 & -1.11419 & 0.55708 \\ 0.00002 & -3.88594 & -1.11457 \end{bmatrix}$$

$$= \begin{bmatrix} 6.16442 & 6.97555 & 4.05555 \\ -0.00002 & 4.04253 & 0.91786 \\ -0.00002 & 0.00001 & 0.84270 \end{bmatrix} = R$$

We have started with matrix A and by means of three elementary rotations we have transformed it into matrix R, that is, we got the transformation QA = R. Note that the elements r(2,1), r(3,1) and r(3,2) haven't been reduced to an absolute zero, but their value is a residue due to the round-offs that are unavoidable in the calculation.

Let's now calculate the matrix Q by a multiplication of the elementary transformation matrices. Also note that the order in which this multiplication is perform is relevant to the result.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -0.27562 & -0.96127 \\ 0 & 0.96127 & -0.27562 \end{bmatrix} \cdot \begin{bmatrix} 0.87359 & 0 & 0.48666 \\ 0 & 1 & 0 \\ -0.48666 & 0 & 0.87359 \end{bmatrix} \cdot$$

$$\cdot \begin{bmatrix} 0.37139 & 0.92848 & 0 \\ -0.92848 & 0.37139 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.87359 & 0 & 0.48666 \\ 0.46781 & -0.27562 & -0.83976 \\ 0.13413 & 0.96127 & -0.24078 \end{bmatrix} = Q$$

We are now able tc calculate QB

$$\begin{bmatrix} 0.87359 & 0 & 0.48666 \\ 0.46781 & -0.27562 & -0.83976 \\ 0.13413 & 0.96127 & -0.24078 \end{bmatrix} \cdot \begin{bmatrix} 12 \\ 3 \\ 8 \end{bmatrix} = \begin{bmatrix} 10.21989 \\ -0.56631 \\ -10.59414 \end{bmatrix} = QB$$

Finally we came up with the system represented by a matrix equation that can be easily solved by Back Substitution, RX = QB. Approximating elements r(2,1), r(3,1) and r(3,2) to zero to eliminate the residual values, we can write RX = QB as

$$\begin{bmatrix} 6.16442 & 6.97555 & 4.05555 \\ 0 & 4.04253 & 0.91786 \\ 0 & 0 & 0.84270 \end{bmatrix} \cdot \begin{bmatrix} x(1) \\ x(2) \\ x(3) \end{bmatrix} = \begin{bmatrix} 10.21989 \\ -0.56631 \\ -10.59414 \end{bmatrix}$$

Solving sequentially for x3, x2, and x1, we get

$$x(3) = -12.57166$$
$$x(2) = 2.71432$$
$$x(1)\ 6.85726$$



Note that we can perform the triangularization with the compound matrix A|B, where "|"is an operator that performs matrix concatenation, and that will result on

$$Q(A|B) = (QA)|(QB)| = R|(QB)$$

This way we can save time with the calculations. This is the scheme adopted when the algorithm is mapped into a systolic array.

## APPENDIX D

### PROGRAM INTERACTION FOR ENTERING A NEW PROBLEM

Note: This example presents the construction of a two cells array and is not related to any particular algorithm.

```
DO YOU WANT TO START A NEW JOB OR TO RUN A PREVIOUS ONE ?
ENTER 'N' FOR NEW OR 'P' FOR PREVIOUS
N
DO YOU WANT TO BE ABLE TO FREEZE AUTOMATIC GRAPHICAL
REVIEW BETWEEN CLOCK CYCLES ? 'Y' OR 'N' ?
N
DO YOU WANT TO RECORD THE SESSION FOR LATER PRINTING ?
         ENTER 'Y' OR 'N'
N


YOU WILL BE PROMPTED TO CONSTRUCT A SYSTOLIC ARRAY
KEEP IN MIND THAT ONLY THE NUMBERS YOU WILL ASSIGN
TO THE CELLS WILL BE KEPT AS IDENTIFICATION BY THE
SYSTEM;  LINKS ARE NOT  IDENTIFIED BY  THEMSELVES,
ONLY BY CELLS THEY CONNECT

ENTER TOTAL NUMBER OF CELLS; MAXIMUM IS 23
 2
```

```
NOW YOU MUST SPECIFY THE CELLS' PROCESSING FUNCTION
AND TASK.
WE HAVE PRESENTLY 8 ROUTINES IN OUR BUILT-IN LIBRARY
AND THEY ARE IDENTIFIED 1 TILL 8

ENTER NUMBER OF DIFFERENT ROUTINES YOU WANT TO USE
 2
ENTER IDENTIFICATION NUMBER OF ROUTINE:
      GIVENS W/ SQRT (internal cell)  => 1
      GIVENS W/ SQRT (external cell)  => 2
      BUFFER CELL                     => 3
      INNER PRODUCT STEP              => 4
      GIVENS W/O SQRT (internal cell) => 5
      GIVENS W/O SQRT (external cell) => 6
      BACK_SUBSTITUTION (round cell)  => 7
      DIVIDED DIFFERENCE              => 8
 1
ENTER NUMBER OF CELLS THAT WILL PERFORM
THIS RCUTINE
 1
ENTER IDENTIFICATICN NUMBER FOR EACH CELL;
HIGHEST ID NUMBER IS 23:
ENTER CELL NUMBER  1 ID NUMBER
         2
CK FCR THIS ROUTINE; IET'S SEE ANOTHER
```

```
ENTER IDENTIFICATION NUMBER OF ROUTINE:
      GIVENS W/ SQRT (internal cell)  => 1
      GIVENS W/ SQRT (external cell)  => 2
      BUFFER CELL                     => 3
      INNER PRODUCT STEP              => 4
      GIVENS W/O SQRT (internal cell) => 5
      GIVENS W/O SQRT (external cell) => 6
      BACK_SUBSTITUTION (round cell)  => 7
      DIVIDED DIFFERENCE              => 8
 2
ENTER NUMBER OF CELLS THAT WILL PERFORM
THIS ROUTINE
 1
ENTER IDENTIFICATION NUMBER FOR EACH CELL;
HIGHEST ID NUMBER IS 23:
ENTER CELL NUMBER  1 ID NUMBER
         1
ENTER NUMBER OF CELLS THAT WILL RECEIVE EXTERNAL DATA
 1
ENTER IDENTIFICATION NUMBER FOR EACH CELL; HIGHEST ID NUMBER IS 23
ENTER CELL NUMBER  1 ID NUMBER
 1
ENTER NUMBER OF CELLS THAT WILL OUTPUT DATA TO EXTERNAL WORLD
 0
```

```
OK, NO OUTPUT TO EXTERNAL WORLD

NOW YCU WILL ASSIGN SHAPE AND  POSITION  ON  SCREEN  TO
EACH CELL THAT WILL CONSTITUTE  YOUR  ARRAY.  AVAILABLE
SHAPES  ARE  OCTAGON  AND  SQUARE.  FOR COORDINATES
ASSIGNMENT, THE SCREEN IS DIVIDED IN 64 POSITIONS  LIKE
A CHESSBCARD AND LEFTMOST BOTTOM POSITION IS (X=1,Y=1).
ABCISSA IS  X  AND IT GOES FROM  1 TO 8. ORDINATE IS Y,
AND IT GOES FROM 1 TO 8 ALSO

ENTER ID NUMBER,SHAPE AND POSITION TO YOUR  2 CELLS

ENTER CELL NUMBER  1 ID NUMBER
 1
ENTER SHAPE:
            SQUARE  ==> 1
            OCTAGON ==> 2
  TAKE A CHOICE|
2

ENTER SCREEN COORDINATES:
      ENTER X (integer from 1 to 8)
 3
      ENTER Y (integer from 1 to 8)
 5
OK, NOW FOR NEXT CELI
```

```
ENTER CELL NUMBER  2 ID NUMBER
 2
ENTER SHAPE:
            SQUARE  ==> 1
            OCTAGON ==> 2
  TAKE A CHOICE|
1

ENTER SCREEN COORDINATES:
      ENTER X (integer from 1 to 8)
 4
      ENTER Y (integer from 1 to 8)
 5


NOW  YOU  WILL  BE  ASKED  TO  DEFINE  THE  LINKING
BETWEEN CELLS; ATTENTION TO THE FACT THAT EACH CELL
HAS 4 PORTS FOR INPUT AND THE SAME NUMBER FOR OUTPUT
YOU MUST SPECIFY  FROM  WHICH  CELL/PORT  TO  WHICH
CELL/PORT THE LINKING MUST BE ESTABLISHED

ENTER THE NUMBER OF LINKS YOU DO WANT TO ESTABLISH
 2
ENTER ID NUMBER OF ORIGIN CELL FROM LINK NUMBER    1
 1
ENTER OUTPUT PORT NUMBER FROM ORIGIN CELL; MAXIMUM IS 4
```

```
1
ENTER ID NUMBER OF DESTINATION CELL FROM LINK NUMBER    1
 2
ENTER INPUT PORT NUMBER FROM DESTINATION CELL; MAXIMUM IS 4
1
ENTER ID NUMBER OF ORIGIN CELL FROM LINK NUMBER    2
 1
ENTER OUTPUT PORT NUMBER FROM ORIGIN CELL; MAXIMUM IS 4
2
ENTER ID NUMBER OF DESTINATION CELL FROM LINK NUMBER    2
 2
ENTER INPUT PORT NUMBER FROM DESTINATION CELL; MAXIMUM IS 4
2
LIST OF WORKING NODES AND THEIR DEFINITION BY THE USER:

NODE NUMBER  1:
     INPUT BUFFER:
        PORT 1:
               DATUM =  0.00000
            WAVEFRONT IS BLACK
        PORT 2:
               DATUM =  0.00000
            WAVEFRONT IS BLACK
        PORT 3:
               DATUM =  0.00000
```

```
        WAVEFRONT IS BLACK
    PORT 4:
            DATUM =  0.00000
        WAVEFRONT IS BLACK
CELL MEMORIES:
    MEMORY 4: 0.00000
    OUTPUT BUFFER:
    PORT 1:
            DATUM =  0.00000
          WAVEFRONT IS BLACK
    PORT 2:
            DATUM =  0.00000
          WAVEFRONT IS BLACK
    PORT 3:
            DATUM =  0.00000
          WAVEFRONT IS BLACK
    PORT 4:
            DATUM =  0.00000
          WAVEFRONT IS BLACK
COLOR IS BLACK
PROCESSOR IS GIVENS EXT W/ SQRT
TASK IS IMPORTER
SHAPE IS OCTAGON

SCREEN COORDINATES:
```

```
            X = 3
            Y = 5


NODE NUMBER  2:
     INPUT BUFFER:
        PORT 1:
               DATUM =  0.00000
            WAVEFRONT IS BLACK
        PORT 2:
               DATUM =  0.00000
            WAVEFRONT IS BLACK
        PORT 3:
               DATUM =  0.00000
            WAVEFRONT IS BLACK
        PORT 4:
               DATUM =  0.00000
            WAVEFRONT IS BLACK
     CELL MEMORIES:
        MEMORY 4: 0.00000
        OUTPUT BUFFER:
        PORT 1:
               DATUM =  0.00000
             WAVEFRONT IS BLACK
```

```
      PORT 2:
             DATUM =  0.00000
            WAVEFRONT IS BLACK
      PORT 3:
             DATUM =  0.00000
            WAVEFRONT IS BLACK
      PORT 4:
             DATUM =  0.00000
            WAVEFRONT IS BLACK
   COLOR IS BLACK
   PROCESSOR IS GIVENS INT W/ SQRT
   TASK IS INTERNAL
   SHAPE IS SQUARE

   SCREEN COORDINATES:

         X = 4
         Y = 5


ANYTHING TO CHANGE IN A NODE ? ENTER 'Y' OR 'N'
Y
ENTER ID NUMBER OF WRONG NODE
 2
SELECT ITEM CHANGE CODE FROM MENU:
```

```
        ID NUMBER     ==> 1
        PROCESSOR     ==> 2
        TASK          ==> 3
        ABCISSA (X)   ==> 4
        ORDINATE (Y)  ==> 5
        SHAPE         ==> 6
2
SELECT NEW PROCESSOR CODE

   GIVENS INTERNAL W/  SQRT ==> 1
   GIVENS EXTERNAL W/  SQRT ==> 2
   BUFFER CELL              ==> 3
   INNER PRODUCT STEP       ==> 4
   GIVENS INTERNAL W/C SQRT ==> 5
   GIVENS EXTERNAL W/C SQRT ==> 6
   BACK SUBSTITUTION        ==> 7
   DIVIDED DIFFERENCE       ==> 8
2
OK, CHANGE IS DONE
DO YOU WANT TO SEE RESULT OF CORRECTION ? ENTER 'Y' OR 'N'
Y
NODE NUMBER  1:
     INPUT BUFFER:
        PORT 1:
```

```
            DATUM =  0.00000
         WAVEFRONT IS BLACK
      PORT 2:
            DATUM =  0.00000
         WAVEFRONT IS BLACK
      PORT 3:
            DATUM =  0.00000
         WAVEFRONT IS BLACK
      PORT 4:
            DATUM =  0.00000
         WAVEFRONT IS BLACK
   CELL MEMORIES:
      MEMORY 4: 0.00000
      OUTPUT BUFFER:
      PORT 1:
            DATUM =  0.00000
          WAVEFRONT IS BLACK
      PORT 2:
            DATUM =  0.00000
          WAVEFRONT IS BLACK
      PORT 3:
            DATUM =  0.00000
          WAVEFRONT IS BLACK
      PORT 4:
```

```
                DATUM =  0.00000
              WAVEFRONT IS BLACK
        COLOR IS BLACK
        PROCESSOR IS GIVENS EXT W/ SQRT
        TASK IS IMPORTER
        SHAPE IS OCTAGON

        SCREEN COORDINATES:

              X = 3
              Y = 5


NODE NUMBER  2:
        INPUT BUFFER:
           PORT 1:
                 DATUM =  0.00000
              WAVEFRONT IS BLACK
           PORT 2:
                 DATUM =  0.00000
              WAVEFRONT IS BLACK
           PORT 3:
                 DATUM =  0.00000
              WAVEFRONT IS BLACK
           PORT 4:
                 DATUM =  0.00000
```

```
        WAVEFRONT IS BLACK
CELL MEMORIES:
   MEMORY 4: 0.00000
   OUTPUT BUFFER:
   PORT 1:
          DATUM =  0.00000
         WAVEFRONT IS BLACK
   PORT 2:
          DATUM =  0.00000
         WAVEFRONT IS BLACK
   PORT 3:
          DATUM =  0.00000
         WAVEFRONT IS BLACK
   PORT 4:
          DATUM =  0.00000
         WAVEFRONT IS BLACK
COLOR IS BLACK
PROCESSOR IS GIVENS EXT W/ SQRT
TASK IS INTERNAL
SHAPE IS SQUARE

SCREEN COORDINATES:

       X = 4
       Y = 5
```

```
ANYTHING TO CHANGE IN A NODE ? ENTER 'Y' OR 'N'
N
LIST OF LINKS ( CONNECTED NODE/PORTS ):
     ORIGIN NODE  1 PORT 1 / DESTINATION NODE  2 PORT 1
     ORIGIN NODE  1 PORT 2 / DESTINATION NODE  2 PORT 2

ANYTHING TO CHANGE IN A LINK ? ENTER 'Y' OR 'N'
Y
YOU WILL BE ALLOWED TO INSERT A NEW LINK OR
TO REMOVE AN EXISTING LINK;  IF  YOUR  NEED
REQUIRES BOTH OPERATIONS, DO ONE AT A TIME.

ENTER ID NUMBER OF ORIGIN CELL
 1
ENTER OUTPUT PORT NUMBER FROM ORIGIN CELL
1
ENTER ID NUMBER OF DESTINATION CELL
 2
ENTER INPUT PORT NUMBER AT DESTINATION CELL
1
SELECT YOUR OPERATION CODE:

     REMOVE LINK ==> 1
     INSERT LINK ==> 2
```

```
1
ANY CTHER LINK FIXING ? ENTER 'Y' OR 'N'
Y
YOU WILL EE ALLOWED TC INSERT A NEW LINK OR
TO REMOVE AN EXISTING LINK;  IF  YOUR  NEED
REQUIRES BOTH OPERATICNS, DO ONE AT A TIME.

ENTER ID NUMBER OF ORIGIN CELL
 2
ENTER OUTPUT PORT NUMEER FROM ORIGIN CELL
1
ENTER ID NUMBER OF DESTINATION CELL
 1
ENTER INFUT PORT NUMBER AT DESTINATION CELL
1
SELECT YOUR OPERATION CODE:

     FEMCVE LINK ==> 1
     INSERT LINK ==> 2
2
ANY CTHER LINK FIXING ? ENTER 'Y' OR 'N'
N
LIST OF LINKS ( CONNECTED NODE/PORTS ):
     CRIGIN NODE  1 PCRT 2 / DESTINATION NODE  2 PORT 2
     CRIGIN NODE  2 PORT 1 / DESTINATION NODE  1 PCRT 1
```

```
ANYTHING TO CHANGE IN A LINK ? ENTER 'Y' OR 'N'
N
ENTER NUMBER OF CLOCK CYCLES TO EXECUTE
 1
NOW YOU'LL ENTER INPUTS TO CELL NUMBER  1 AT TIME t= 1
HOW MANY EXTERNAL INPUT PORTS ARE ACTIVE?
1
 1 EXTERNAL INPUT PORTS ? 'Y' OR 'N' ?
1
HOW MANY EXTERNAL INPUT PORTS ARE ACTIVE?
1
 1 EXTERNAL INPUT PORTS ? 'Y' OR 'N' ?
Y
ENTER ID NUMBER OF PORT 1
1
PORT ID IS  1 ? 'Y' OR 'N' ?
Y
ENTER INPUT DATUM TO PORT ID NUMBER 1
    5.000
INPUT DATUM IS     5.000 ? 'Y' OR 'N' ?
Y
ENTER COLOR CODE FOR THIS DATUM WAVEFRONT :
        BLACK ==> 1
        BLUE  ==> 2
```

```
        RED   ==> 3
        GREEN ==> 4
3
WAVEFRCNT COLORCODE IS 3 ? 'Y' OR 'N' ?
Y
LIST OF CELLS AND THEIR TRANSITORY STATUS:

NODE NUMBER  1:
     INPUT BUFFER:
        PORT 1:
               DATUM =  5.00000
            WAVEFRONT IS RED
        PORT 2:
               DATUM =  0.00000
            WAVEFRONT IS BLACK
        PORT 3:
               DATUM =  0.00000
            WAVEFRONT IS BLACK
        PORT 4:
               DATUM =  0.00000
            WAVEFRONT IS BLACK
     CELL MEMORIES:
        MEMORY 4: 0.00000
        OUTPUT BUFFER:
        PORT 1:
```

```
                DATUM =  0.00000
              WAVEFRONT IS RED
         PORT 2:
                DATUM =  1.00000
              WAVEFRONT IS RED
         PORT 3:
                DATUM =  0.00000
              WAVEFRONT IS BLACK
         PORT 4:
                DATUM =  0.00000
              WAVEFRONT IS BLACK
      CCLCR IS RED
      PROCESSOR IS GIVENS EXT W/ SQRT
      TASK IS IMPORTER
      SHAPE IS OCTAGON

      SCREEN COORDINATES:

            X = 3
            Y = 5


NODE NUMBER  2:
      INPUT BUFFER:
         PORT 1:
                DATUM =  0.00000
```

```
        WAVEFRONT IS BLACK
    PORT 2:
           DATUM =  0.00000
        WAVEFRONT IS BLACK
    PORT 3:
           DATUM =  0.00000
        WAVEFRONT IS BLACK
    PORT 4:
           DATUM =  0.00000
        WAVEFRONT IS BLACK
CELL MEMORIES:
    MEMORY 4: 0.00000
    OUTPUT BUFFER:
    PORT 1:
           DATUM =  1.00000
         WAVEFRONT IS BLACK
    PORT 2:
           DATUM =  0.00000
         WAVEFRONT IS BLACK
    PORT 3:
           DATUM =  0.00000
         WAVEFRONT IS BLACK
    PORT 4:
           DATUM =  0.00000
```

```
          WAVEFRCNT IS BLACK
     COICR IS BLACK
     PROCESSOR IS GIVENS EXT W/ SQRT
     TASK IS INTERNAI
     SHAPE IS SQUARE

     SCREEN COORDINATES:

          X = 4
          Y = 5


DO YOU WANT TO REVIEW THE WHOLE SEQUENCE ? ENTER 'Y' OR 'N'
Y
START OF PRESENTATION
LOOK AT YOUR  DIRECTORY  FOR  FILE  SYSARRAY.MEM; IT
CONTAINS A RECORDING OF THE PROCESSING FOR  GRAPHICS
REVIEW; IF YOU ALSO ASKED FOR RECORDING FOR PRINTING
THERE IS ANOTHER RECORD CALLED SYSPRINT.DOC THAT YOU
CAN ASK THE OPERATING SYSTEM TO SEND TO  PRINTER  IF
YOU WISH;

        THIS IS THE END OF THE SESSION
```

## APPENDIX E

### PROGRAM INTERACTION FOR REVIEW OF A PREVIOUS SESSION

```
DO YOU WANT TO START A NEW JOB OR TO RUN A PREVIOUS ONE ?
ENTER 'N' FOR NEW OR 'P' FOR PREVIOUS
P
DO YOU WANT TO BE ABLE TO FREEZE AUTOMATIC GRAPHICAL
REVIEW BETWEEN CLOCK CYCLES ? 'Y' OR 'N' ?
Y
DO YOU WANT TO RECORD THE SESSION FOR LATER PRINTING ?
         ENTER 'Y' OR 'N'
N

JUST A MOMENT
START OF PRESENTATION
START NEXT CLOCK CYCLE ? 'Y' OR 'N' ?
Y
START NEXT CLOCK CYCLE ? 'Y' OR 'N' ?
Y
START NEXT CLOCK CYCLE ? 'Y' OR 'N' ?
Y
START NEXT CLOCK CYCLE ? 'Y' OR 'N' ?
Y
LOOK AT YOUR  DIRECTORY  FOR  FILE  SYSARRAY.MEM; IT
CONTAINS A RECORDING OF THE PROCESSING FOR  GRAPHICS
```

REVIEW; IF YOU ALSO ASKED FOR RECORDING FOR PRINTING THERE IS ANOTHER RECORD CALLED SYSPRINT.DOC THAT YOU CAN ASK THE OPERATING SYSTEM TO SEND TO PRINTER IF YOU WISH;

THIS IS THE END OF THE SESSION

## LIST OF REFERENCES


1. Kung, H.T. and Leiserson, C.E., Systolic Arrays (for VLSI), Carnegie Melon University, Department of Computer Science, 1978.

2. Kung, H.T., "Why Systolic Architectures?", IEEE Computer, January 1982.

3. Speiser, J.M. and Whitehouse, H.J., "A Review of Signal Processing with Systolic Arrays", Real Time Signal Processing IV, Vol. 431, pages 23-25, 1983.

4. NOSC Technical Document 588, NOSC Systolic Processor Testbed, by J.J. Symanski, 1983.

5. Weiser, U. and Davis, Al, A Wavefront Notation Tool for VLSI Array Design, Carnegie Melon University Conference on VLSI Systems and Computations, 1981.

6. Rice, J.R., Matrix Computations and Mathematical Software MacGraw-Hill Book Company, 1981.

7. Gentleman, W.M. and Kung, H.T., "Matrix Triangularization by Systolic Arrays", Proceedings of SPIE, Real Time Signal Processing IV, Vol. 298, 1982.

8. Gentleman, W.M., "Least Squares Computations by Givens Transformations without Square Roots", J.Inst.Maths.Applics., Vol. 12, 1973.

9. Tewarson, R.P., Sparse Matrices, Academic Press, 1973.

10. Li, Tao and Smith, Kent F., "An Equilateral-Triangular Systolic Architecture for Computing Divided Differences", Proceedings IEEE, International Conference on Computer Design, 1984.

11. Kung H.T. and Iam, Monica S., "Fault-Tolerance and Two-Level Pipelining in VLSI Systolic Arrays", Proceedings of the Conference on Advanced Research in VLSI, MIT, January 1984.

12. Hohn, F.E., Elementary Matrix Algebra, The Mcmillan Company, 1973.

# BIBLIOGRAPHY

ACM/SIGGRAPH Graphics Standards Planning Comittee, Wenner, P. and Henry, L., Reference Manual and Functional Specification of the Core Computer Graphics System, George Washington University, 1981.

Dan, A. Van and Foley, J.D. Fundamentals of Interactive Computer Graphics, Addison-Wesley Publishing Company, 1983.

Forsythe, G. and Moler, C.B., Computer Solution of Linear Algebraic Systems, Prentice-Hall Inc., 1967.

Hammarling, Sven, "A Note on Modifications to the Givens Plane Rotation", J.Inst.Maths.Applics., Vol. 13, 1974.

McWhirter,J.G., "Recursive Least Squares Minimization Using a Systolic Array", Proceedings of SPIE, Real Time Signal Processing IV, Vol. 298, 1982.

Westlake, J.R., A Handbook of Numerical Matrix Inversion and Solution of Linear Systems, John Wiley & Sons, 1968.

## INITIAL DISTRIBUTION LIST

| No. | | Copies |
|---|---|---|
| 1. | Defense Technical Information Center<br>Cameron Station<br>Alexandria, Virginia 22304-6145 | 2 |
| 2. | Library, Code 0142<br>Naval Postgraduate School<br>Monterey, California 93943-5100 | 2 |
| 3. | Superintendent<br>ATTN: Prof. Harriet Rigas, Code 62Rr<br>Naval Postgraduate School<br>Monterey, California 93943-5100 | 1 |
| 4. | Superintendent<br>ATTN: Prof. Robert Denney Strum, Code 62St<br>Naval Postgraduate School<br>Monterey, California 93943-5100 | 1 |
| 5. | Superintendent<br>ATTN: Prof. Donald Evan Kirk, Code 62Ki<br>Naval Postgraduate School<br>Monterey, California 93943-5100 | 1 |
| 6. | Superintendent<br>ATTN: Prof. Chin-Hwa Lee, Code 62Le<br>Naval Postgraduate School<br>Monterey, California 93943-5100 | 5 |
| 7. | Superintendent<br>ATTN: Atsuko Keiyu, Code 62<br>Naval Postgraduate School<br>Monterey, California 93943-5100 | 1 |
| 8. | Ed Holland, Code 811<br>Naval Ocean System Center<br>San Diego, California 92152 | 1 |
| 9. | Rich O'Connel, Code 811<br>Naval Ocean System Center<br>San Diego, California 92152 | 1 |
| 10. | Mike Stelmach, Code 811<br>Naval Ocean System Center<br>San Diego, California 92152 | 1 |
| 11. | LCDR Leopoldo Jorge de Souza (Brazilian Navy)<br>Brazilian Naval Commission<br>4706 Wiscosin Avenue N.W.<br>Washington D.C. 20016 | 5 |

21305

Thesis
S66623
c.1

Souza
Algorithmic study on systolic array structures.